A Linguagem de Programação C

1. Conceitos Básicos

1.1. Linguagens de Programação

"Uma linguagem é um conjunto sistemático de regras para comunicar idéias, como as linguagens naturais". Uma Linguagem de Programação (LP) difere das Linguagens Naturais (LN) em várias formas:

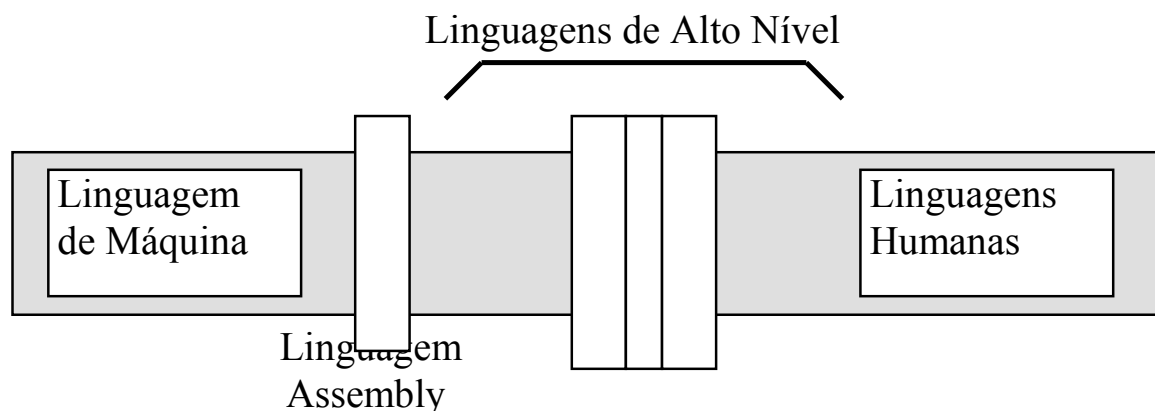
Linguagem Natural	Linguagem de Programação
Pessoa➔pessoas	Pessoa➔computador
Conteúdo genérico	Conteúdo=programa
Meio usado: voz	Meio usado: cadeias de caracteres

"Uma linguagem de programação é uma linguagem utilizada por uma pessoa para expressar um processo através do qual o computador possa resolver um problema". Os que possuem uma compreensão limitada da LN são limitados na complexidade de expressar seus pensamentos, especialmente em termos de capacidade de abstração.

Programadores que tiveram pouca educação formal em Ciência da Computação tendem a continuar a usar a mesma LP, mesmo que esta esteja em desuso. A aprendizagem contínua é fundamental. É necessário que os programadores conheçam os fundamentos das LPs para que possam ler, entender e aprender com os manuais e livros técnicos das novas LPs e os novos paradigmas que venham a surgir.

1.2. Implementação de linguagens de programação

O computador é uma máquina capaz de seguir uma espécie de algoritmo chamado programa que está escrito em linguagem de máquina. Os 2 principais componentes de um computador são a memória interna e o processador. A memória é usada para armazenar programas e dados.

O processador é um conjunto de circuitos que garante a realização de operações primitivas – instruções de máquina ou macroinstruções – que formam sua linguagem de máquina (LM).

O sistema operacional (SO) e as implementações das LPs são dispostos em camadas sobre a interface da LM de um computador. Essas camadas podem ser imaginadas como computadores virtuais que oferecem interfaces em nível mais alto para o usuário, conforme pode ser visto na figura 1.2:

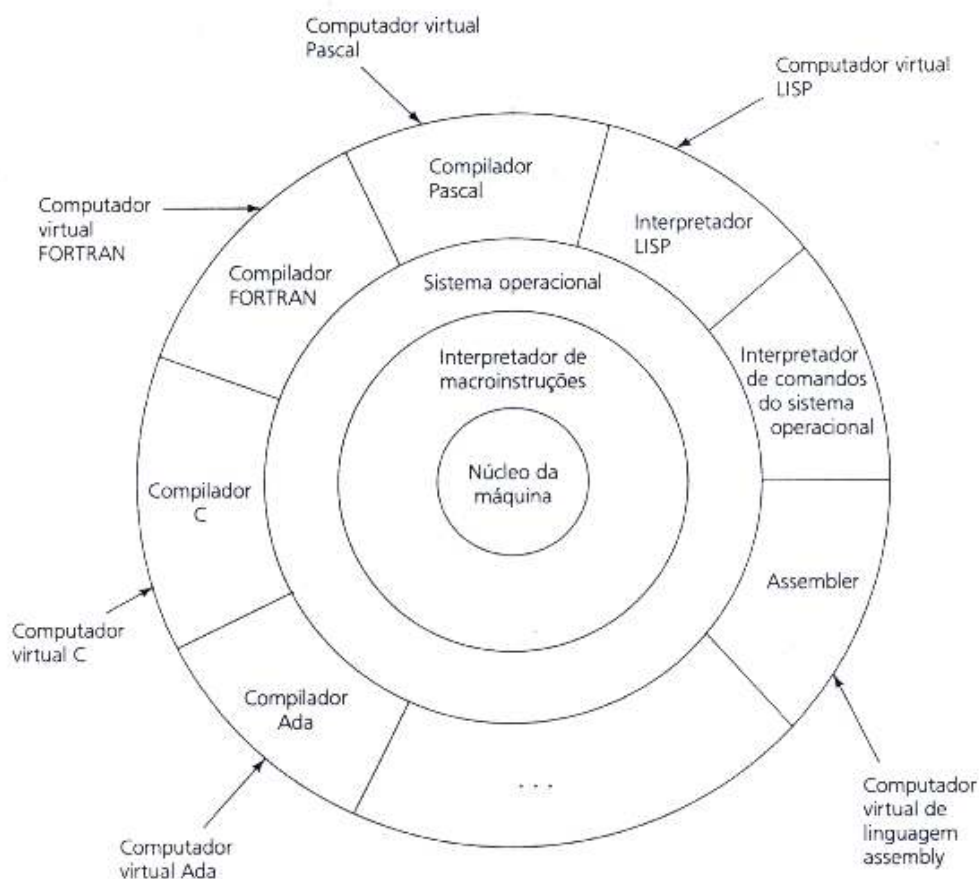



Figura 1.2 – Implementações de linguagens de programação

Todo programa escrito em uma LP deve ser traduzido para a linguagem de máquina para ser executado. Isso é realizado através de um programa ou conjunto de programas. Esse programa tradutor recebe como entrada o código fonte e gera o código de máquina. Existem três maneiras de se fazer a tradução:

- **Compilação**
- **Interpretação**
- **Híbrido**

1.2.1. Compilação

Efetua a tradução integral do código fonte para o código de máquina. A execução é mais rápida porque não é necessário fazer nenhuma tradução intermediária.

Para que o programa seja executado é necessário apenas o código executável. A desvantagem é a não portabilidade do código executável.

Não há depuração, pois o código executável não guarda referência do código fonte.

O processo de compilação se desenvolve em diversas etapas, que podem ser vistas na figura 1.2.1.

Programa-fonte
Analisador léxico
Unidades léxicas
Analisador sintático
Árvores de análise (*parse trees*)
Tabela de símbolos
Gerador de código intermediário (e analisador sintático)
Otimização
(opcional)
Código intermediário
Gerador de código
Linguagem de máquina
Dados de entrada
Computador
Resultados

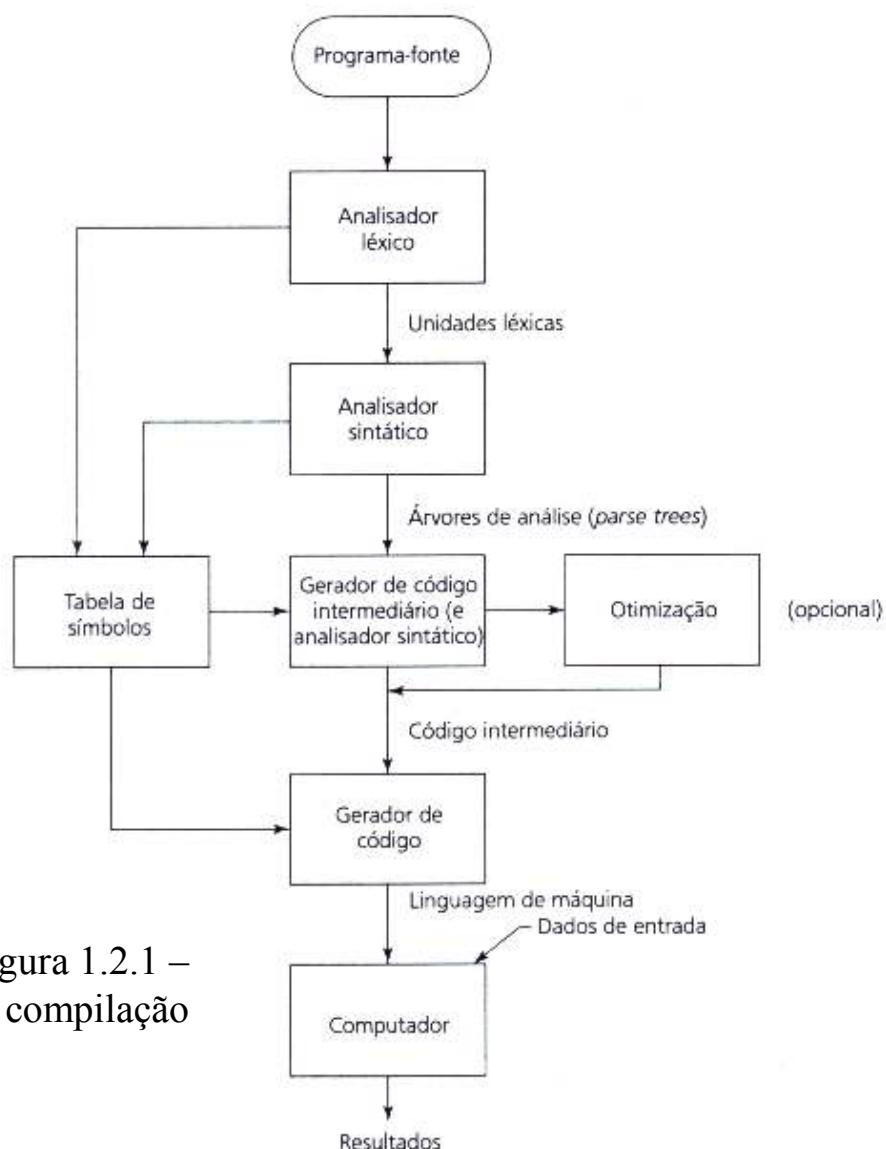



Figura 1.2.1 –
O processo de compilação

- O analisador léxico agrupa os caracteres em identificadores, palavras reservadas, símbolos e operadores.
- O analisador sintático monta a estrutura sintática a partir das unidades léxicas.
- O gerador de código intermediário produz um programa em uma linguagem que se assemelha ou é a própria linguagem de montagem (*assembly*).
- A otimização (opcional) melhora o programa tornando-o menor e mais rápido.
- O gerador de código converte a versão do código intermediário otimizado para LM.
- A tabela de símbolos contém as informações sobre tipos e atributos de cada nome definido pelo programador.

1.2.2. Interpretação

O programa interpretador "decifra" as instruções escritas em uma LP. No momento da execução, a instrução é traduzida para linguagem de máquina e executada, conforme mostra a figura 1.2.2.

A vantagem é que apenas partes do programa podem ser executados, mas há desvantagens: o processo é mais lento em relação ao processo de compilação e há maior consumo de memória, pois há a necessidade de armazenar o código-fonte, tabela de símbolos e o programa interpretado durante a interpretação.

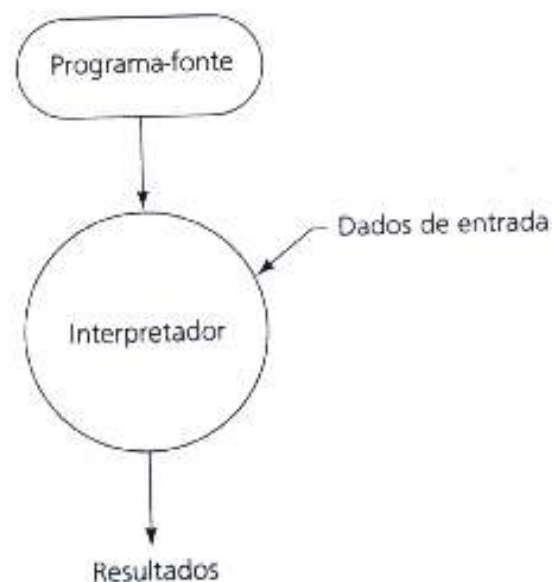



Figura 1.2.2 – O processo de interpretação

1.2.3. Híbrido

O processo híbrido combina a execução eficiente e a portabilidade de programas. A base é a existência de um código intermediário, mais fácil de ser interpretado e não específico de uma plataforma computacional.

O método é dividido em duas etapas: compilação para um código intermediário e interpretação desse código, conforme mostra a figura 1.2.3.

A linguagem PERL é implementada como um sistema híbrido, assim como as implementações iniciais do JAVA.

O código intermediário do Java, chamado bytecode ou código de bytes, oferece portabilidade para qualquer máquina que tenha um interpretador e um sistema de *run-time* associado, conhecido como Java Virtual Machine.

Há sistemas que traduzem o código de bytes para linguagem de máquina para permitir uma execução mais rápida.

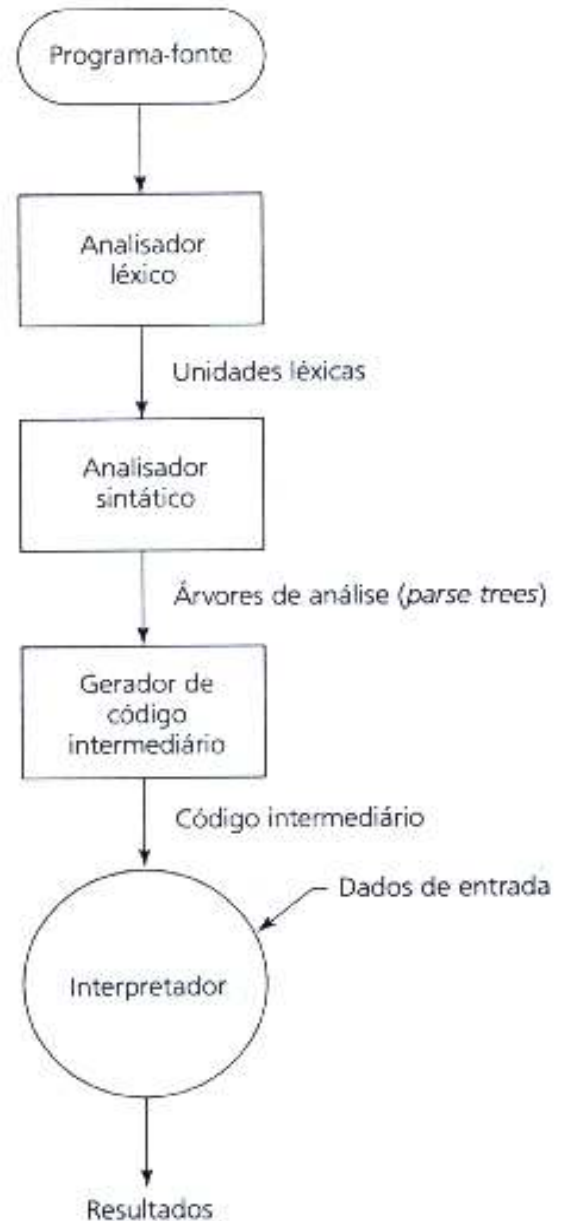



Figura 1.2.3 –
O processo híbrido

1.3. Histórico do desenvovimento da linguagem C

- O processo de desenvolvimento da linguagem C:

- Iniciado com a criação da **linguagem BCP** (Martin Richards).
- Seguiu-se a criação da **linguagem B** (Ken Thompson).
- **Linguagem C**: criada em 1972 por Dennis Ritchie, nos Laboratórios da Bell Telephone como uma linguagem para escrever sistemas operacionais. **Objetivo**: projetar o sistema Unix. (Primeira vez que uma linguagem de alto nível foi designada especificamente para o desenvolvimento de sistemas).

- Inicialmente usada apenas em sistemas UNIX.
- Única especificação formal: **"The C Reference Manual"** por Ritchie. 1977: Ritchie e Brian Kerninghan expandiram para o livro: **"The C Programming Language"**.

- 1983: **Padrão ANSI** (*American National Standards Institute*). Objetivo de garantir portabilidade para diferentes ambientes.

1.4. Características Gerais da Linguagem C

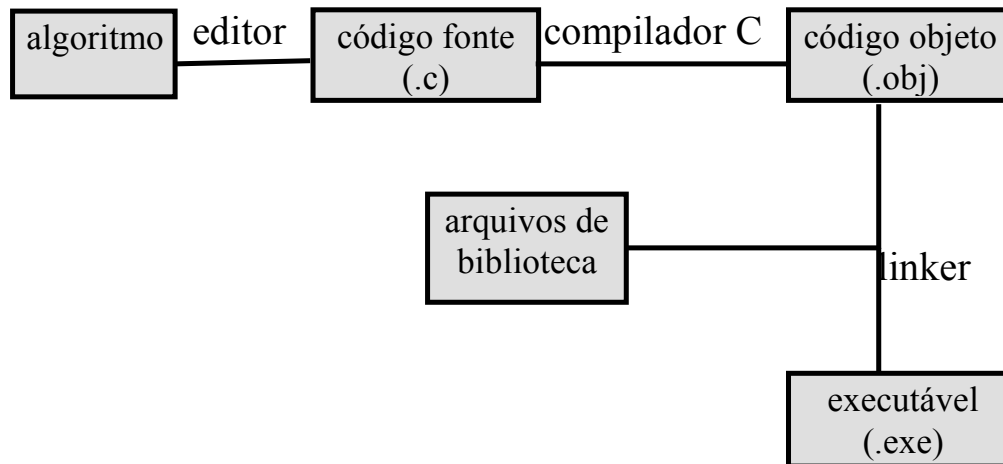
- Permite ao programador definir estruturas de dados construídas com base nos seus dados primitivos;
- Diferencia entre letras maiúsculas e minúsculas;
- Permite recursividade de funções;
- É uma linguagem de nível médio: permite a manipulação de bits, bytes e endereços (ponteiros);
- Não possui comandos de I/O (read e write) - devem-se usar funções específicas para realizar estas tarefas.

- Qualidades:
  - Uso abrangente: - Sistemas Operacionais, Processadores de texto
    - Programas Gráficos, Planilhas
    - Compiladores
  - Altamente portável;
  - Utiliza poucas palavras-chave (32 = 27 padrão KR + 5 ANSI);
  - Modular;

Tabela 1.1. Lista das palavras-chave do ANSI C

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

" *C retém a filosofia básica de que os programadores sabem o que estão fazendo* "
(em **C - A Linguagem de Programação Padrão ANSI,**
de Brian W. Kernighan e Dennis M. Ritche)

- **Processo de Compilação:**

- Algoritmo: estrutura do programa
- Editor de texto: alguns compiladores C possuem ambiente de programação integrados, incluindo o editor de textos
- Código fonte: conjunto de comandos da linguagem C, escrito em ASCII, que o programador expressa sua lógica de programação [extensão .c]
- Compilador C: a linguagem C é compilada
- Código objeto: arquivo intermediário, está em linguagem de montagem (*assembly*) [extensão .obj]
- Arquivos de biblioteca: contém funções já compiladas
- Linker: cria um programa executável a partir de arquivos objeto e dos arquivos de bibiloteca
- Executável: programa C que pode ser executado no computador, está em linguagem de máquina [extensão .exe]

- **Biblioteca de Linkedição**

- Coleção de arquivos objeto;
- Biblioteca padrão mínima (padrão ANSI).

Funções divididas em grupos:
  - Entrada/Saída;
  - Gerenciamento de memória;
  - Operações matemáticas;
  - Manipulação de cadeias de caracteres.

- Biblioteca fornecida pelo compilador:
  - Funções gráficas

1.5. Estrutura de um programa em C

- Componentes iniciais:

Comandos de pré-processamento:

- Inclusão de arquivos de cabeçalho (da biblioteca);
- Definição de constantes.

Declaração de variáveis, estruturas e ponteiros globais.

Inicialização de variáveis, constantes, estruturas e ponteiros globais.

Declaração de protótipos de funções: fornece ao compilador o nome e os argumentos das funções contidas no programa.

- Função main:

Função especial por onde o programa inicia sua execução.

Contem as componentes essenciais com as instruções que comandam toda a execução do programa.

- Funções:

Desempenham tarefas específicas no programa.

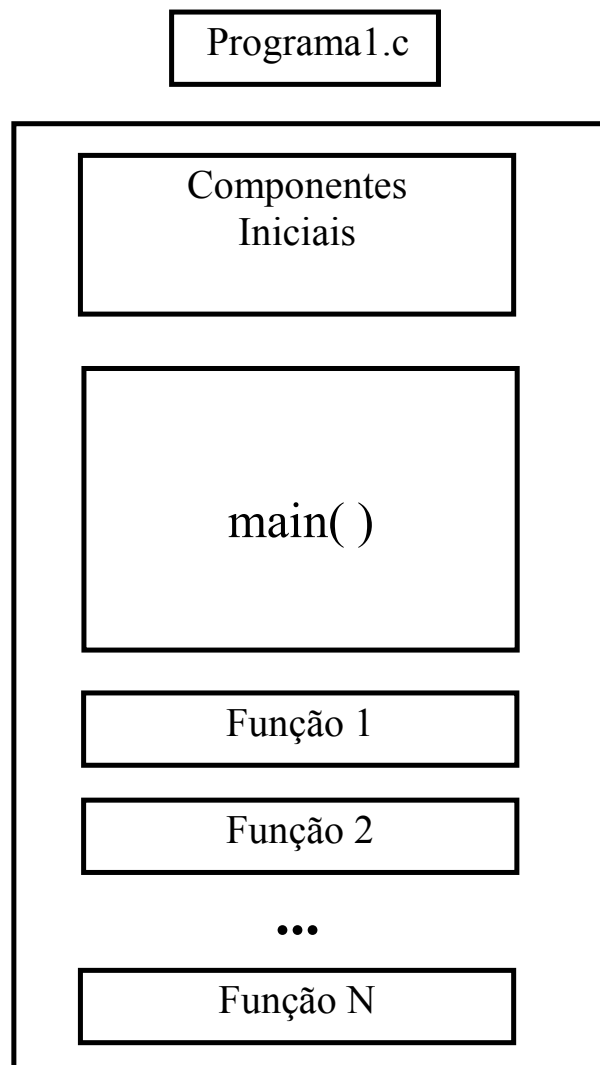
Programa1.c

Componentes Iniciais

main( )

Função 1

Função 2

...

Função N

Exemplo 1.1:

```
// Media.c - calcula a média aritmética de dois números inteiros

#include <stdio.h>

int a,b;       // declara a e b como variáveis inteiras
float m;       // declara m como variável de ponto flutuante

float media(int x, int y); /* declara media como uma função que recebe
                            2 números inteiros e retorna um inteiro */
```

```
void main(void)
{
// Solicita o primeiro valor usando a função de biblioteca printf
    printf("Qual o primeiro valor ? ");

// Lê o primeiro valor da tela usando a função de biblioteca scanf
    scanf("%i",&a);

// Solicita o segundo valor usando a função de biblioteca printf
    printf("Qual o segundo valor ? ");

// Lê o segundo valor da tela usando a função de biblioteca scanf
    scanf("%i",&b);

    m=media(a,b);     /*Chama a função média para calcular a
                          média entre a e b e atribui o resultado a m*/

// Mostra na tela o resultado
    printf("\nA média é %.2f: ",m);
}
```

```
float media(int x, int y)
{  float result;

   result = (float) (x+y) /2;
   return (result);
}
```

1.6. Comentários em C

- O hábito de se introduzir alguns comentários em um programa é bastante salutar. Estes devem ser bem colocados, de tal forma a facilitar a manutenção do programa:

- Esclarecendo passagens que não sejam muito óbvias;
- Elucidando o papel de uma variável;
- Contando resumidamente como funciona uma função.

entre outras...

- Comentários em C devem iniciar com um **/*** e terminar com ***/**, podendo começar em uma linha e terminar em outra. Ou usa-se // no início da linha.

Exemplos:

```
//Isto é um comentario
/* Isto
   é outro comentário */
```

Lista de Exercícios Nº 1

1- Retire a linha #include <stdio.h> do programa MEDIA.C e tente criar um executável. Qual a mensagem de erro? De que tipo ela é? O programa executa? Anote-a.

2- Retire a letra a da linha int a,b; do programa MEDIA.C e tente criar um executável. Qual a mensagem de erro? Anote-a.

3- Retire a linha float media (int x, int y); do programa MEDIA.C e tente criar um executável. Qual a mensagem de erro? Anote-a.

4- Retire o ; da linha m=media(a,b); do programa MEDIA.C e tente criar um executável. Qual a mensagem de erro? Anote-a.

5- Troque m por M na linha m=media(a,b); no programa MEDIA.C e tente criar um executável. Qual a mensagem de erro? Anote-a.

6- Retire o / da linha /* Chama a função ... no programa MEDIA.C e tente criar um executável. Qual a mensagem de erro? Anote-a.*

7- Escreva uma rotina main( ) que imprima Bom dia!

8- Encontre os erros no uso das funções:

```
scanf(“Quantos erros tem este programa?, cont_de_problemas )
printf( Este argumento tem %d erros, cont_de_problemas );
```

2. Instruções de Entrada e Saída

Conjunto de funções que permitem a entrada de dados no programa e a saída de resultados.

2.1. A função `printf( )`

- Arquivo de cabeçalho a ser incluído: **stdio.h**
- <stdio.h>: define tipos e macros necessários para o pacote padrão de Entrada/Saída definido por K.&R. e declara as funções desse pacote;
- Imprime dados na tela;

Sintaxe:

```
printf ("string_de_formato", itens_de_dados);
```

- Primeiro argumento: string_de_formato → deve estar entre aspas ("") e pode conter:
  - ♦ Texto;
  - ♦ Códigos de barra invertida;
  - ♦ Especificadores de formato:

%[largura][.precisão]tipo

- Segundo em diante: itens_de_dados, tantos quantos forem os especificadores de formato.

Exemplo:

```
      .
      .
float f1 = 1.2;
int   i2 = 1;
char  c3 = 'a';
printf("Imprimindo três valores: %f, %d, %c", f1, i2, c3 );
```

Tabela 2.1 Especificadores de Formato

Código	*Formato*
`%c`	Caracter único.
`%s`	Cadeia de caracteres.
`%d %i (%ld)`	Inteiro decimal (longo).
`%u`	Inteiro decimal não sinalizado.
`%f (%lf)`	Ponto flutuante (double).
`%e (%E)`	Ponto flutuante em notação científica com e minúsculo (maiúsculo).
`%g (%G)`	Usa %e (%E) ou %f (%F), o que for mais curto.
`%o`	Inteiro octal.
`%x (%X) (%lx)`	Inteiro hexadecimal com letras em minúsculo (maiúsculo) (longo).
`%p`	Apresenta um ponteiro.
`%n`	O argumento associado é um ponteiro para inteiro, no qual o número de caracteres escritos até esse ponto é colocado.
`%%`	Escreve o símbolo %.

2.1.1. Mais sobre a função `printf( )`

O modificador #:

- Precedendo-se os especificadores de formato **g**, **f** e **e** com # garante-se que haverá um ponto decimal, mesmo que não haja dígitos decimais.

- Se o especificador de formato **x** é precedido por um #, o número hexadecimal será escrito com um prefixo **0x**.

- O modificador # não pode ser aplicado a nenhum outro especificador de formato.

O modificador *:

- Os especificadores de largura mínima de campo e precisão podem ser fornecidos como argumentos de **printf()**, em lugar de constantes: usa-se, para tanto, o * como substituto.

Exemplo:

Os modificadores **l** e **h**:

- Podem ser aplicados aos especificadores de formato **d, i, o, u** e **x**.

- O modificador **l** diz a **printf()** que segue um tipo de dado **long**.

Exemplo:

```
printf("%ld\n",longint); // um long int será mostrado
```

-O modificador **h** diz a **printf()** que segue um tipo de dado **short**.

Exemplo:

```
printf("%hu\n",long);     /* um short unsigned int
                             será mostrado */
```

O modificador **L**:

- Pode anteceder o especificador de ponto flutuante **e**, **f** ou **g** e indica que segue um **long double**.

Justificando a saída:

- Para se justificar a saída à esquerda (o default é justificar à direita), coloca-se um sinal de menos imediatamente após o %.

Exemplo:

```
float f1= 12.1;
printf("%-8.3f\n", f1); /* Justifica à esquerda um número em
                           ponto flutuante com 3 casas decimais
                           e um campo de 8 caracteres */
```

O especificador de precisão:

- O especificador de precisão aplicado a um dado em ponto flutuante, determina o número de casas decimais mostrado.

Exemplo:

```
float f1= 12.112131415;
printf("%.3f\n", f1); /* Especifica que 3 casas decimais devam
                         ser mostradas */
```

- Quando aplicado a um dado inteiro, determina o número mínimo de dígitos que aparecerão, sendo que zeros iniciais são adicionados para completar o número de dígitos solicitado.

- Quando aplicado a strings, determina o comprimento máximo do campo. Se a string for maior que a largura máxima, os caracteres serão truncados.

Exemplo:

```
printf("%3.5s\n", string); /* Mostra uma string de pelo menos
                              3 e não excedendo 5 caracteres */
```

Exemplo 2.1:

```
/* Printf.c - Ilustra vários especificadores de formato da função printf,
   seu uso com especificadores de largura e precisão bem como modificadores
   e justificação */

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

#define ESPECIFICADOR_G                         1
#define ESPECIFICADOR_E                         2
#define ESPECIFICADORES_IOX                     3
#define ESPECIFICADOR_N                         4
#define ESPECIFICADOR_P                         5
#define ESPECIFICADOR_LARGURAEPRECISAO          6
#define MODIFICADOR                             7
#define JUSTIFICACAO                            8

void menu(int opcao);

void main()
{  int opcao;

   for(;;)
   {  do
      { puts("\nEscolha uma das opcoes do menu:\n");
        puts("1. Ilustrando os especificadores de formato %%g e %%G.\n");
        puts("2. Ilustrando os especificadores de formato %%e e %%E.\n");
        puts("3. Ilustrando os especificadores de formato %%i, %%o e %%x.\n");
        puts("4. Ilustrando os especificadores de formato %%n.\n");
        puts("5. Ilustrando os especificadores de formato %%p.\n");
        puts("6. Ilustrando os especificadores de largura e precisao.\n");
        puts("7. Ilustrando o modificador *.\n");
        puts("8. Ilustrando justificacao a direita e a esquerda.\n");
        puts("9. Sair.\n");
        scanf("%d", &opcao);
      }while( opcao < 1 || opcao > 9);
      if (opcao == 9)
         exit(0);
      menu(opcao);
   }
}

void menu(int opcao)
{  double f1;
   double f2=11.121314;
   float  f3=10;
   unsigned numero;
   int ondemoro;
   int conta;
```

```
switch(opcao)
{ case ESPECIFICADOR_G:
      printf("Ilustrando o uso dos especificadores de formato %%g e %%G\n");
      printf("%%g\t\t%%G\n");
      for(f1=1.0; f1<1.0e+10; f1=f1*10)
       printf("%g\t%G\n", f1,f1);
      break;

   case ESPECIFICADOR_E:
       printf("Ilustrando o uso dos especificadores de formato %%e e %%E\n");
       printf("%%e\t\t%%E\n");
       for(f1=1.0; f1>1.0e-10; f1=f1/10)
           printf("%e\t%E\n", f1,f1);
      break;

    case ESPECIFICADORES_IOX:
         printf("Ilustrando o uso do especificador de formato %%i, %%o e %%x\n");
         printf("%%i\t%%o\t%%x\n");
         for(numero=0; numero<50; numero++)
        { if (!(numero%21) && numero!=0)
          { printf("Aperte qualquer tecla para continuar...\n");
             getch();
             printf("%%i\t%%o\t%%x\n");
           }
           printf("%d\t%o\t%x\n", numero, numero, numero);
         }
         break;

    case ESPECIFICADOR_N:
         printf("\n\nIlustrando o uso do especificador de formato %%n\n");
         printf("Com o %%n na posicao abaixo:\n");
         printf("printf("\nVejamos%%n quantas letras ha antes de vejamos.",\
                   &conta);\n\n");
         printf("Vejamos%n  quantas letras ha antes de vejamos.\n",&conta);
         printf("\nconta assume o valor: %d que e igual a quantidade de letras\
               da palavra Vejamos\n",conta);
         break;

   case ESPECIFICADOR_P:
         printf("\n\nIlustrando o uso do especificador de formato %%p\n");
         printf("para mostrar o endereco na maquina da variavel ondemoro.\n");
         printf("O endereco de ondemoro e:%p\n", &ondemoro);
         break;

   case ESPECIFICADOR_LARGURAEPRECISAO:
         printf("\n\nIlustrando o uso de especificadores de largura minima\
               de campo\n");
         printf("\nUsando:    %%f para formatar f2, obtem-se f2=%f", f2);
         printf("\nUsando:  %%14f para formatar f2, obtem-se f2=%14f", f2);
         printf("\nUsando: %%014f para formatar f2, obtem-se f2=%014f", f2);
         printf("\n\nO especificador de precisao tem efeitos diferentes\n");
         printf("quando aplicado a um inteiro, a um float ou a uma string\n");
         printf("Aplicado ao float f2=11.121314 como %%.2f resulta:\n");
         printf("%.2f\n",f2);
          conta=7;
         printf("Aplicado ao int conta=7 como %%3.5d resulta:\n");
         printf("%3.5d\n",conta);
         printf("Aplicado a uma string como \"Eu vou, eu vou, pra casa agora\
                eu vou!" \n como %%10.20, resulta:\n");
         printf("%10.20s\n","Eu vou, eu vou, pra casa agora eu vou!");
         break;
```

```
        case MODIFICADOR:
           printf("\n\nIlustrando o uso do modificador *.\n");
           printf("Usando * para fornecer os especificadores de largura minima\
     e precisao\n como argumentos para printf.Seja f2=11.121314, o comando:\n");
           printf("printf(\"%%*.*f\",14,2,f2)\n");
           printf("Resulta:\n");
           printf("%*.*f\n",14,2,f2);
           break;

        case JUSTIFICACAO:
           printf("Ilustrando justificacao a direita e a esquerda\n");
           printf("Justificando f3=10 com %%10f a direita:\n");
           printf("%20f\n", f3);
           printf("Justificando f3 com %%-10f a esquerda:\n");
           printf("%-20f\n", f3);
           break;
      }
      printf("Pressione qualquer tecla para continuar...\n");
      getch();
  }
```

Exemplo 2.2:

```
/* Escapes.c - Mostra o resultado de algumas seqüências de escape*/

#include <stdio.h>
#include <conio.h>

void main()
{ printf("Ilustrando o uso de sequencias de escape\n");
  printf("Emitindo sinais sonoros:\a\a\a\a\a");
  puts("\n\nPressione uma tecla para continuar...");
  getch();

  printf("\n\nComo funciona o caracter de nova linha. \n");
  printf("Obtem-se com: \n printf(\"3 novas linhas: \\n\\n\\n, certo!\")\n");
  printf("3 novas linhas: \n\n\n, certo!");
  puts("\n\nPressione uma tecla para continuar...");
  getch();

  printf("\n\nComo funciona o caracter de retrocesso. \n");
  printf("Obtem-se com: \n printf(\"5 retrocessos: \\b\\b\\b\\b\\b, certo!\")\n");
  printf("5 retrocessos: \b\b\b\b\b, certo!");
  puts("\n\nPressione uma tecla para continuar...");
  getch();
  printf("\n\nComo funciona o caracter de retorno de carro. \n");
  printf("Obtem-se com: \n printf(\"3 retornos de carro: \\r\\r\\r, certo!\")\n");
  printf("3 retornos de carro: \r\r\r, certo!");
  puts("\n\nPressione uma tecla para continuar...");
  getch();
  printf("\n\nComo funciona o caracter de tabulacao horizontal. \n");
  printf("Obtem-se com: \n printf(\"3 tabulacoes horizontais: \\t\\t\\t, certo!\")\n");
  printf("3 tabulacoes horizontais: \t\t\t, certo!");
  puts("\n\n Fim do programa...");
  getch();
}
```

2.2. Variante da função printf( ): `sprintf( )`

- É idêntica à função **printf()** exceto pelo fato de que a saída, ao invés de ir para a tela, é colocada em uma matriz caracter, que é o primeiro argumento da função.

Exemplo:

```
char str[80];
sprintf(str, "%d %s", 80, "ção");   /*str conterá "80 ção" */
```

2.3. A função `scanf( )`

- Arquivo de cabeçalho a ser incluído: **stdio.h** (biblioteca entrada/saída);
- Lê dados fornecidos pelo teclado;

Sintaxe:

```
scanf ("cadeia_de_formato", itens_de_dados);
```

- Pode receber qualquer número de argumentos:
  - O primeiro argumento é uma cadeia de formato; os especificadores de formato são os mesmos usados com a função **printf**;
  - Os argumentos de itens de dados numéricos devem ser precedidos pelo operador **& (endereço de).**

Tabela 2.2. Códigos de barra invertida

Código	*Significado*
`\a`	Sinal sonoro
`\b`	Retrocesso
`\f`	Alimentação de formulário
`\n`	Nova linha
`\r`	Retorno de carro
`\t`	Tabulação horizontal
`\v`	Tabulação vertical
`\0`	Nulo
`\\`	Barra invertida
`\?`	Ponto de interrogação
`\' ( \" )`	Aspas simples (dupla)

2.3.1. Mais sobre a função `scanf( )`

Lendo cadeias de caracteres:

- A **`scanf()`** pode ser usada para a leitura de cadeias de caracteres, usando-se o especificador de formato **%s**.
- Para **`scanf()`**, um caracter de espaço em branco é um espaço, um retorno de carro ou uma tabulação. Deste modo, os caracteres da cadeia são lidos até que um espaço em branco seja encontrado, portanto **`scanf()`** não pode ser utilizada para ler uma string como: "Isto é um teste", porque o primeiro espaço encerra o processo de leitura.

Deve-se passar endereços para **`scanf()`**

- As variáveis utilizadas para receber valores através de **`scanf()`**, devem ser passadas pelo seu endereço, isto é precedidas pelo operador **&.**
- As strings são lidas para matrizes de caracteres e o nome da matriz sem qualquer índice é o endereço do primeiro elemento da mesma, portanto, neste caso, o operador **&** não deve ser usado.

Exemplo:

```
scanf("%s", str); /*str já é um ponteiro*/
```

Descartando espaços em branco indesejados

- Um caracter de espaço em branco na string de controle faz com que **`scanf()`** salte um ou mais caracteres de espaço em branco que ocorram na entrada.

Exemplo:

```
int n1, n2;
scanf("%d %d",&n1,&n2); /*Use scanf com o espaço em branco
                          entre os %d para ler os números:1  2 */
```

Caracteres diferentes de "espaço em branco" na string de controle

- Um caracter diferente de espaço em branco na string de controle faz com que **`scanf`** leia e ignore caracteres iguais na entrada.

Exemplo:

```
scanf("%d,%d",&n1,&n2); /* faz com que scanf() leia um
   inteiro, leia e descarte uma vírgula e leia, então, outro inteiro */
```

- Se o caracter não for encontrado, **`scanf()`** termina.

Uso de especificadores de largura máxima

- Ao se especificar uma largura máxima para uma string, apenas o número de caracteres especificado será lido.

Exemplo:

```
    scanf("%20s", str);    /* Apenas os 20 primeiros caracteres
                                 serão passados para str */
```

- Se a string, no exemplo anterior, tiver mais que 20 caracteres, uma chamada subseqüente a qualquer função de entrada de caracteres começará onde esta terminou.

Utilizando um "Scanset"

- Um scanset define um conjunto de caracteres que pode ser lido pela **scanf()**.

- É definido colocando-se uma string dos caracteres a serem procurados entre colchetes. Também pode-se especificar uma faixa de caracteres, usando-se um hífen entre o caracter inicial e o final.

Exemplos:

```
char str1[80], str2[80];

scanf("%d %[XYZ] %s",&i,str1,str2);

/*Após ler  um inteiro, que é atribuido a i irá procurar por
  uma seqüência contendo as letras X Y e/ou Z, atribuindo-a a
  str1 e, quando aparecer um caracter que não seja nenhum destes
  três, passa a atribuir o restante a str2*/

scanf("%d %[A-Z] %s",&i,str1,str2);

/* Para str1 procurará por uma seqüência contendo letras de
   A a Z (mas não de a a z, pois letras minúsculas e maiúsculas
   são discriminadas)*/
```

Exemplo 2.3:

```
/* Scanf.c - Mostra o uso da funçao scanf com scansets, descartes de
             caracteres e especificadores de largura máxima */
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#define SCANSET                   1
#define DESCARTE_CARACTERES      2
#define ESPECIFICADOR_LARGURA    3

void menu(int opcao);

void main()
{     int opcao;
      for(;;)
      { do
        { puts("\nEscolha uma das opcoes do menu:\n");
           puts("1. Ilustrando scanset.\n");
           puts("2. Ilustrando o descarte de virgulas e outros caracteres.\n");
           puts("3. Ilustrando o uso de especificador de largura.\n");
           puts("4. Sair.\n");
           scanf("%d", &opcao);
        } while( opcao < 1 || opcao > 4);
        if (opcao == 4)
             exit(0);
         menu(opcao);
        }
}
void  menu(int opcao)
{   int i,i1;
    float f,f1;
    char str1[80], str2[80];
    char str[80];
    switch(opcao)
    {case SCANSET:
        printf("Sugestao: Forneca uma sequencia iniciada com um numero inteiro,\n\
              seguido de caracteres contendo algumas das letras:a,b,c ou d\n");
        scanf("%d%[abcd]%s", &i, str1, str2);
        printf("\nO numero e: %d\n", i);
        printf("\nOs caracteres do scanset sao: %s\n", str1);
        printf("\nOs caracteres restantes sao: %s\n", str2);
       printf("\nSugestao: Forneca uma sequencia iniciada com um numero\n\
              em ponto flutuante seguido de caracteres contendo \n\
              algumas das letras entre A e Z\n");
        scanf("%f%[A-Z]%s", &f, str1, str2);
        printf("\nO numero e: %f\n", f);
        printf("\nOs caracteres do scanset sao: %s\n", str1);
        printf("\nOs caracteres restantes sao: %s\n", str2);
        break;

     case ESPECIFICADOR_LARGURA:
       printf("\n\nIlustrando o uso de especificadores de largura\n");
       printf("Forneca uma string sem espacos em branco com mais de 5
             caracteres:\n");
       scanf("%5s",str1);
       printf("\nComo o especificador de largura e 5, so foi atribuido a str1:%s\n",str1);
       scanf("%s",str2);
       printf("\nE numa chamada subsequente a scanf com str2, foi atribuido a este:
             %s\n",str2);
       break;
     case DESCARTE_CARACTERES:
       printf("\n\nIlustrando o descarte de virgulas e outros caracteres\n");
       printf("Usando:\n scanf(\"%%d,;%%f\\n\",&i, &f);\n");
       printf("Forneca um numero inteiro seguido por um float,\n\
              separados por uma virgula e um ponto e virgula.\n\
              Termine com os caracteres ^z e pressione <enter>:");
        scanf("%d,;%f\n",&i, &f);
        printf("Os numeros fornecidos foram %d e %f\n", i, f);
       gets(str);
        printf("\n Agora usando:\n scanf(\"%%d%%f\\n\",&i, &f);\n");
       printf("Forneca os mesmos numeros,\n\ separados por um ponto e virgula.\n\
              Termine com os caracteres ^z e pressione <enter>:");
       scanf("%d;%f\n",&i1, &f1);
       printf("Os numeros fornecidos foram %d e %f\n", i1, f1);
       gets(str);      /* para pegar o lixo que sobra */
       break;
   }
printf("Acione qualquer tecla para continuar\n");
getch()
}
```

2.4. A função **puts( )**

- Arquivo de cabeçalho a ser incluído: **stdio.h**
- Permite exibir textos na tela, mas não exibe valores de variáveis.
- Ao final de uma mensagem, esta função automaticamente acrescenta um caracter de mudança de linha.
- Reconhece os mesmos códigos de barra invertida de **printf()**.

Exemplo:

```
puts("Tempo bom com pouca nebulosidade");
```

2.5. A função **gets( )**

- Arquivo de cabeçalho a ser incluído: **stdio.h**
- Lê uma string de caracteres fornecida pelo teclado e coloca-a no endereço apontado por seu argumento ponteiro de caracteres.
- Podem-se digitar caracteres até que seja pressionado o retorno de carro, quando então **gets()** retorna.
- Quando pressionada a tecla <Enter> é acrescentado um terminador nulo ao final da string.
- Podem-se corrigir erros de digitação usando a tecla <backspace>.

Exemplo:

```
char str[80];
puts("Digite seu endereço completo ");
gets(str);
```

2.6. A função **getchar( )**

- Arquivo de cabeçalho a ser incluído: **stdio.h.**
- Não recebe argumentos.
- Espera até que uma tecla seja pressionada e devolve o seu valor.
- A tecla é automaticamente mostrada na tela.
- Devolve um inteiro, mas é apenas o byte de mais baixa ordem que contém o caracter.

2.7. A função **putchar( )**

- Arquivo de cabeçalho a ser incluído: **stdio.h.**
- Recebe um argumento do tipo int, mas também pode ser chamada com um argumento do tipo caracter.

- Escreve seu argumento na tela a partir da posição corrente do cursor.
- Devolve o caracter escrito ou, na ocorrência de um erro, devolve EOF (definido em stdio.h geralmente como -1)

2.8. As funções `getch( )` e `getche( )`

- Arquivo de cabeçalho a ser incluído: **conio.h.**
- <conio.h>: declara várias funções usadas para chamar rotinas de entrada/saída pelo console.
- Lêem um único caracter do teclado.
- Ambas esperam até que uma tecla seja pressionada e então retornam imediatamente.
- **getch()** não ecoa o que foi digitado na tela.
- **getche()** ecoa o que foi digitado na tela.

Sintaxe:

```
variável_caracter = getche();
variável_caracter = getch();
```

Exemplo 2.4:

```
/* Getput.c: Ilustra o uso de algumas funcões de entrada/saída */
#include <stdio.h>
#include <conio.h>

void main()
{      char str_nome[80];
       char str_idade[10];
       char ch;
        puts ("Ilustra a utilizacao de getch getche puts gets ");
       puts("\nQual e o seu nome?");
       gets(str_nome);
       puts("\nQuantos anos voce tem? (Pode mentir...)");
       gets(str_idade);
       puts("\nQual o seu endereco? Termine-o com um ponto (.):\n");
       do
       {      ch = getch();
              putchar(ch);
       } while(ch!='.');

       puts("\n\nQual o seu telefone? Termine-o com um ponto (.):\n");
       do
       { ch = getche();
       } while(ch!='.');
       printf("\n\nTchau. Tchau.\n");
}
```

Lista de Exercícios Nº 2

1- Escreva e compile um programa que leia uma variável do tipo int, outra do tipo float, outra do tipo char e uma string, todas separadas por vírgula e mostre-as na tela.

2. Escreva e compile um programa que receba uma variável do tipo float e duas variáveis do tipo int interpretadas como largura e precisão e imprima na tela a variável do tipo float com estes valores de largura e precisão duas vezes, uma justificada à direita e outra à esquerda.

3- Escreva e compile um programa tal que dado um número entre 0 e 255, apresente as seguintes opções:

- *Mostrar o símbolo ASCII correspondente ao número;*
- *Mostrar o símbolo ASCII correspondente ao número que sucede aquele fornecido;*
- *Mostrar o símbolo ASCII correspondente ao número que antecede aquele fornecido;*
- *Sair do programa.*

Use as funções puts() e getch() onde achar que pode.

3. Tipos de Dados

Dados: são informações, que podem ser números ou caracteres, com as quais o programa opera.

3.1. Variáveis

Representam localizações da memória onde são armazenados valores que podem ser modificados pelo programa.

- Nomes de variáveis:

- Podem conter letras, números e o caracter de sublinhado. *Exemplo:* `BR_01`
- O primeiro caracter deve ser uma letra ou caracter de sublinhado.
- Letras maiúsculas e minúsculas correspondem a variáveis diferentes. *Exemplo:* `A1, a1`
- Não podem ser usadas palavras reservadas.

- Tipos básicos:

- **`char`**
- **`int`**
- **`float`**
- **`double`**
- **`void`**

- Modificadores de tipo:

- **`signed`**
- **`unsigned`**
- **`long`**
- **`short`**

- O tamanho (em bytes) e a faixa destes tipos variam em função do processador e da implementação do compilador

- O padrão ANSI estabelece uma faixa mínima, mostrada na Tabela 3.1.

Tabela 3.1. Tipos de Dados Definidos no Padrão ANSI

Tipo	*Palavra Chave*	*Tamanho aprox. em bytes*	*Faixa mínima*
caracter	`char`	1	-127 a 127
caracter não sinalizado	`unsigned char`	1	0 a 255
caracter sinalizado	`signed char`	1	-127 a 127
inteiro	`int`	2	-32.767 a 32.767
inteiro não sinalizado	`unsigned int`	2	0 a 65535
inteiro sinalizado	`signed int`	2	-32.767 a 32.767
inteiro curto	`short int`	2	-32.767 a 32.767
inteiro curto não sinalizado	`unsigned short int`	1	0 a 65535
inteiro curto sinalizado	`signed short int`	1	-32.767 a 32.767
inteiro longo	`long int`	4	-2.147.483.647 a 2.147.483.647
inteiro longo sinalizado	`signed short int`	4	-2.147.483.647 a 2.147.483.647
inteiro longo não sinalizado	`unsigned short int`	4	0 a 4.294.967.295
ponto flutuante com precisão simples	`float`	4	Seis dígitos de precisão
ponto flutuante com precisão dupla	`double`	6	Dez dígitos de precisão
ponto flutuante com precisão dupla longo	`long double`	8	Dez dígitos de precisão

- Declaração de variáveis

- todas as variáveis devem ser declaradas antes de serem usadas

Sintaxe:

```
tipo lista de variáveis;
```

Exemplo:

```
int x,y;
float a;
```

- podem ser declaradas:
  - ◊ dentro de funções: variáveis locais;
  - ◊ fora de quaisquer funções: variáveis globais;
  - ◊ como parâmetros formais de funções.

Exemplo:

```
float media(float x, float y)
```

- Inicialização de variáveis

- pode-se atribuir um valor a uma variável ao declará-la ou em outra parte do programa onde é permitido o uso de comandos

  Exemplo:
```
long var=0;

     ou

long var;
void main(void)
{   var=0;
     ...
}
```

3.1.1. O operador em tempo de compilação "`sizeof`"

- Operador unário que retorna o tamanho em bytes da variável ou especificador de tipo que ele precede

Exemplo:
```
float f;
    printf( "%d", sizeof f );
    printf( "\n%d", sizeof(int));
```

Obs: O nome de um tipo deve ser colocado entre parênteses. Isto não é necessário para nomes de variáveis.

Exemplo 3.1:

```
/* Sizeof.c - Programa que verifica o tamanho de cada variável
   no computador que se está usando */

#include <stdio.h>

int main( )
{
printf("\nNeste computador uma variável char tem %d byte(s)",sizeof(char));
printf("\nNeste computador uma variável int tem %d bytes",sizeof(int));
printf("\nNeste computador uma variável short tem %d byte(s)",sizeof(short));
printf("\nNeste computador uma variável long tem %d bytes",sizeof(long));
printf("\nNeste computador uma variável unsigned char tem %d byte(s)",\
          sizeof(unsigned char));
printf("\nNeste computador uma variável unsigned int tem %d bytes",\
          sizeof(unsigned int));
printf("\nNeste computador uma variável unsigned long tem %d bytes",\
           sizeof(unsigned long));
printf("\nNeste computador uma variável float tem %d bytes",sizeof(float));
printf("\nNeste computador uma variável double tem %d bytes",\
          sizeof(double));
return 0;
}
```

3.2. Contantes

Representam localizações na memória de dados que não podem ser alterados durante a execução do programa.

- Tipos de constantes: qualquer um dos cinco tipos de dados básicos.

- Por default o compilador encaixa uma constante numérica no menor tipo de dado compatível que pode contê-la.

  Exemplo: `20` constante do tipo **`int`**

  `100000` constante do tipo **`long`**

- Exceção à regra do menor tipo: constantes em ponto flutuante são assumidas sempre como **`double`**.

  Exemplo:

  $$\left.\begin{array}{l} 20. \\ 1.23E2 \\ 1.2E-2 \end{array}\right\} \rightarrow$$ constantes do tipo **`double`**

- Pode-se especificar o tipo com o uso de sufixos.

Tabela 3.2. Sufixos para a Especificação de Constantes

Tipo de dado	*Sufixo*	*Exemplos de Constantes*
`int`	-	`1 123 21000 -234`
`long int`	L	`35000L -34L`
`short int`	-	`10 -12 90`
`unsigned int`	U	`10000U 987U`
`float`	F	`123.23F 4.34e-3F`
`double`	-	`123.23 -0.9876324`
`long double`	L	`1001.2L`

- Constante caracter: envolvida por ‘ ’

  Exemplo: ‘a’, ‘%’

- Constante string: conjunto de caracteres colocados entre aspas “ ” (Atenção: Não confundir strings com caracteres)

  Exemplo: `'a', "Bom dia!"`

- Constante hexadecimal (base 16): deve consistir de um `0x` seguido por uma constante na forma hexadecimal.

  Exemplo: `0x80`

- Constante octal (base 8): deve consitir de um `0` seguido de uma constante na forma octal.
  Exemplo: `012`

- Declaração de constantes:

- Constantes simbólicas:

 - São constantes representadas por um nome. Ao defini-las deve-se atribuir a elas um valor.
 - Uma constante simbólica é definida usando-se a diretiva **#define**:

Sintaxe:

```
#define NOME_DA_CONSTANTE  valor
```

> **Convenção:**
> nome de CONSTANTES - letras maiúsculas
> nome de variáveis - letras minúsculas

Exemplo:

```
#define PI 3.1416
#define LETRA 'a'
```

- Uso do modificador de tipo **const**:

Sintaxe:

```
const tipo_da_constante NOME = valor;
```

Exemplo:

```
const int ALTURA = 30;
const float CRITERIO = 0.0008;
const char LETRA= 'a';
```

Exemplo 3.2:

```
/* Altura.c - Transforma a altura de uma pessoa de metros para centímetros. */

#include <stdio.h>

/* define uma constante de conversão de metros para centímetros */
/* 1metro = 100 cm */

#define CONVERTE 100

float h_metros, h_cm;

main( )
{  /* recebe os dados do usuário */
   printf("Qual a sua altura em metros? ");
   scanf("%f",&h_metros);

   /* faz a conversão */
   h_cm = CONVERTE* h_metros;

   /* exibe o resultado */
  printf("\nA sua altura em centímetros é %.2f cm', h_cm);
  return 0;
}
```

Exemplo 3.3:

```
/* Const.c - Programa que mostra como declarar constantes usando sufixos
             e exibe o tamanho em bytes destas constantes */
#include <stdio.h>
#define INT          35
#define LONG        35L
#define FLOAT       123.0F
#define DOUBLE      123.0
#define OCTAL       012
#define HEXA   0x80
#define LONGHEXA    0X3A7C2FB5L

void main()
{
  const char primeira = 'V', do_meio = 'u';
  const char ultima[] = "s";
  const char primeira_metade[] = "amos ver alg";
  const char segunda_metade[] = "mas constante";

  printf("%c%s%c%s%s:\n",primeira,primeira_metade,do_meio,segunda_metade,ultima);
  printf( "O tamanho em bytes da constante INT= %d e: %d\n", INT, sizeof INT);
  printf( "O tamanho em bytes da constante LONG= %ld e: %d\n", LONG, sizeof LONG);
  printf( "O tamanho em bytes da constante FLOAT= %f e: %d\n", FLOAT, sizeof FLOAT);
  printf( "O tamanho em bytes da constante DOUBLE= %f e: %d\n", DOUBLE,\
         sizeof DOUBLE);
  printf( "O tamanho em bytes da constante OCTAL= %o e: %d\n", OCTAL, sizeof OCTAL);
  printf( "O tamanho em bytes da constante HEXA= %x e: %d\n", HEXA, sizeof HEXA);
  printf( "O tamanho em bytes da constante LONGHEXA= %lx e: %d\n", LONGHEXA,\
         sizeof LONGHEXA);
}
```

3.2.1. Definição de conjuntos de constantes através de “enum”

- A palavra-chave **enum**: define uma enumeração, que consiste em um conjunto de constantes inteiras, nomeadas, denominadas numeradores.

Sintaxe:

```
    enum nome { lista de enumeração }
```

Exemplo:

```
enum meses {janeiro=1, fevereiro, março, abril,
            maio, junho, julho, agosto, setembro,
            outubro, novembro, dezembro };

int mes = setembro;
```

- Se a nenhum enumerador for atribuído um valor, os valores das constantes correspondentes começam em 0 e aumentam de 1 à medida que a declaração é lida da esquerda para a direita.

Exemplo 3.4:

```
/* Enum.c - Programa que mostra como definir constantes
            usando a palavra-chave enum */

#include <stdio.h>

enum moeda{real, dollar, euro, libra};
char *nome_moeda[] = {"real", "dollar", "euro","libra"};

void main()
{    int i;

     for( i = real; i<=libra; i++ )
          printf("%s\n",nome_moeda[i]);

     printf("O real é representado pelo número: %d\n", real);
}
```

3.3. O tipo de dados `char`

Uma constante caracter é um inteiro e a cada caracter corresponde um valor numérico da tabela de códigos ASCII. Assim, 97 é o código ASCII para a letra **a**. Pode-se imprimir, através de um programa, o valor do código, **97**, ou o símbolo associado a ele, **a**.

Exemplo 3.5

```
   /* Codigos.c - Fornece o código ASCII de
      uma tecla acionada */

#include <stdio.h> /* Por causa da funcao printf */
#include <conio.h> /* Por causa da funcao getche */

#define FIM 26  /* Código ASCII de ^Z */

void main()
{    int c;
     do
     {    c=getche();
          printf("  Tecla: %d\n",c);
     } while(c != FIM );
}
```

Lista de Exercícios Nº 3

1- *Altere o programa do exemplo 2.2 para usar const ao invés de #define.*

2- *Escreva e compile um programa que converta o peso de Kg para gramas.*

3- *Escreva e compile um programa que calcula o salário de uma pessoa em dólares.*

4- *No programa enum.c tente atribuir um valor qualquer a uma das constantes: real, dollar, etc. O que ocorre? Anote.*

5- *Escreva um programa que enumere os meses do ano e imprima:*
Janeiro é o mês 1
Fevereiro é o mês 2
.
.

Tabela 3.3. Valores ASCII – *American Standard Code for Information Interchange*

<table>
<tr><th>Char</th><th>Dec</th><th>Char</th><th>Dec</th><th>Char</th><th>Dec</th><th>Char</th><th>Dec</th></tr>
<tr><td>NUL ^@</td><td>0</td><td>SPACE</td><td>32</td><td>@</td><td>64</td><td>‘</td><td>96</td></tr>
<tr><td>^A</td><td>1</td><td>!</td><td>33</td><td>A</td><td>65</td><td>a</td><td>97</td></tr>
<tr><td>^B</td><td>2</td><td>“</td><td>34</td><td>B</td><td>66</td><td>b</td><td>98</td></tr>
<tr><td>^C</td><td>3</td><td>#</td><td>35</td><td>C</td><td>67</td><td>c</td><td>99</td></tr>
<tr><td>^D</td><td>4</td><td>$</td><td>36</td><td>D</td><td>68</td><td>d</td><td>100</td></tr>
<tr><td>^F</td><td>5</td><td>%</td><td>37</td><td>E</td><td>69</td><td>e</td><td>101</td></tr>
<tr><td>^F</td><td>6</td><td>&</td><td>38</td><td>F</td><td>70</td><td>f</td><td>102</td></tr>
<tr><td>^G</td><td>7</td><td>‘</td><td>39</td><td>G</td><td>71</td><td>g</td><td>103</td></tr>
<tr><td>^H</td><td>8</td><td>(</td><td>40</td><td>H</td><td>72</td><td>h</td><td>104</td></tr>
<tr><td>TAB ^I</td><td>9</td><td>)</td><td>41</td><td>I</td><td>73</td><td>i</td><td>105</td></tr>
<tr><td>^J</td><td>10</td><td>*</td><td>42</td><td>J</td><td>74</td><td>j</td><td>106</td></tr>
<tr><td>^K</td><td>11</td><td>+</td><td>43</td><td>K</td><td>75</td><td>k</td><td>107</td></tr>
<tr><td>^L</td><td>12</td><td>,</td><td>44</td><td>L</td><td>76</td><td>l</td><td>108</td></tr>
<tr><td>^M</td><td>13</td><td>-</td><td>45</td><td>M</td><td>77</td><td>m</td><td>109</td></tr>
<tr><td>^N</td><td>14</td><td>.</td><td>46</td><td>N</td><td>78</td><td>n</td><td>110</td></tr>
<tr><td>^O</td><td>15</td><td>/</td><td>47</td><td>O</td><td>79</td><td>o</td><td>111</td></tr>
<tr><td>^P</td><td>16</td><td>0</td><td>48</td><td>P</td><td>80</td><td>p</td><td>112</td></tr>
<tr><td>^Q</td><td>17</td><td>1</td><td>49</td><td>Q</td><td>81</td><td>q</td><td>113</td></tr>
<tr><td>^R</td><td>18</td><td>2</td><td>50</td><td>R</td><td>82</td><td>r</td><td>114</td></tr>
<tr><td>^S</td><td>19</td><td>3</td><td>51</td><td>S</td><td>83</td><td>s</td><td>115</td></tr>
<tr><td>^T</td><td>20</td><td>4</td><td>52</td><td>T</td><td>84</td><td>t</td><td>116</td></tr>
<tr><td>^U</td><td>21</td><td>5</td><td>53</td><td>U</td><td>85</td><td>u</td><td>117</td></tr>
<tr><td>^V</td><td>22</td><td>6</td><td>54</td><td>V</td><td>86</td><td>v</td><td>118</td></tr>
<tr><td>^W</td><td>23</td><td>7</td><td>55</td><td>W</td><td>87</td><td>w</td><td>119</td></tr>
<tr><td>^X</td><td>24</td><td>8</td><td>56</td><td>X</td><td>88</td><td>x</td><td>120</td></tr>
<tr><td>^Y</td><td>25</td><td>9</td><td>57</td><td>Y</td><td>89</td><td>y</td><td>121</td></tr>
<tr><td>^Z</td><td>26</td><td>:</td><td>58</td><td>Z</td><td>90</td><td>z</td><td>122</td></tr>
<tr><td>^[</td><td>27</td><td>;</td><td>59</td><td>[</td><td>91</td><td>{</td><td>123</td></tr>
<tr><td>^|</td><td>28</td><td><</td><td>60</td><td>\</td><td>92</td><td>|</td><td>124</td></tr>
<tr><td>^]</td><td>29</td><td>=</td><td>61</td><td>]</td><td>93</td><td>}</td><td>125</td></tr>
<tr><td>^^</td><td>30</td><td>></td><td>62</td><td>^</td><td>94</td><td>~</td><td>126</td></tr>
<tr><td>^_</td><td>31</td><td>?</td><td>63</td><td>_</td><td>95</td><td>del</td><td>127</td></tr>
</table>

4. Operadores

- Fazem com que uma determinada operação, ou ação, seja executada com um ou mais operandos.
- Há quatro classes principais de operadores:

Aritméticos
Lógicos
Relacionais
Bit a bit

- Há outros operadores que não se encaixam na classificação acima:

- Operador de atribuição e de atribuição compostos
- Operador condicional
- Operador vírgula
- Operador `sizeof` (seção 3.1.1)
- Operador de endereçamento e operador de deferenciação

(serão vistos em conjunção com ponteiros)

- Operador ponto e operador seta (serão vistos em conjunção com estruturas e com ponteiros)
- Operador molde
- Operador parênteses e colchetes

4.1. Operador de Atribuição

- Sintaxe:

```
nome_da_variável = expressão;
```

- Pode ser usado dentro de qualquer expressão válida em C

Exemplo:

```
x=4*var+3;          /*Atribui o valor da expressão a x*/
y=2;                /*Atribui o valor 2 a y*/
```

4.1.1. Conversão de tipos em atribuições

- Ocorre quando variáveis de um tipo são misturadas com variáveis de outro tipo.
- Regra de Conversão: O valor do lado direito de uma atribuição é convertido no tipo do lado esquerdo.

<u>Exemplo:</u>

```
int x;
char ch;
float f;
```

`ch = x;` => Os bits mais significativos da variável inteira **x** são ignorados, deixando **ch** com os bits menos significativos. Se **x** está entre 0 e 255, então **ch** e **x** têm valores idênticos.

`x = f;` => **x** recebe a perte inteira de **f**.

`f = ch;` => **f** converte o valor inteiro de 8 bits armazenado em **ch** no mesmo valor em formato de ponto flutuante.

`f = x;` => **f** converte um valor inteiro de 16 bits no formato de ponto flutuante.

Exemplo 4.1:

```
/* Converte.c:Mostra a ocorrência de conversão de tipos em atribuições */

#include <stdio.h>
void main()
{ int x;
  char ch;
  float f;

x = 122;        ch = x;
printf("O valor de x e: %d\n", x);
printf("O valor de ch = x é: %d\n", ch);

x=300;               ch = x;
printf("\nO valor de x e: %d\n", x);
printf("O valor de ch = x é: %d\n", ch);

ch = 'a';       f = ch;
printf("\nO valor de ch é: %d, correspondendo ao caracter: %c\n",ch,ch);
printf("O valor de f = ch é: %f\n", f);

f = 98.8999;         ch = f;
printf("\nO valor de f é: %f\n", f);
printf("O valor de ch=f é: %d, correspondendo ao caracter: %c\n", ch,ch);

f = x;
printf("\nO valor de x é: %d\n", x);
printf("O valor de f = x é: %f\n", f);
}
```

- <u>Atribuições múltiplas:</u> é possível atribuir o mesmo valor a muitas variáveis em um mesmo comando.

Exemplo:

```
int x, y, z;
x = y = z = 0;
```

==> **Cuidado!**

```
float f;
int x;
x = f = 3.5;  // Resulta em f=3.5 e x=3
f = x = 3.5;  // Resulta em f=3.0 e x=3
```

4.2. Operadores Aritméticos

- **Operadores unários:** Atuam sobre apenas um operando

Tabela 4.1. Operadores Unários

Operador	*Ação*
`- (menos unário)`	Multiplica seu único operando por -1
`++ (incremento)`	Incrementa o operando em uma unidade
`-- (decremento)`	Decrementa o operando em uma unidade

Exemplos:

```
x=x+1;      →      ++x;
x=x-1;      →      --x;
```

- ++x, incrementa o valor de x antes de usá-lo **(modo pré-fixado)**
- x++, incrementa o valor de x depois de usá-lo **(modo pós-fixado)**
- --x, decrementa o valor de x antes de usá-lo **(modo pré-fixado)**
- x--, decrementa o valor de x depois de usá-lo **(modo pós-fixado)**

Exemplo 4.2:

```
/* Unarios.c - Exemplifica o uso dos operadores unários. */

#include<stdio.h>

void main()
{ int a,b,a1,b1;
  int j,k,m,n;

  a=b=a1=b1=5;
  j=k=0;
  m=n=1;

  printf("\nValores iniciais: a=%d e a1=%d",a,a1);
  printf("\na++=%d e a1--=%d",a++,a1--);
  printf("\na=%d e a1=%d",a,a1);
  printf("\nValores iniciais: b=%d e b1=%d",b,b1);
  printf("\n++b=%d e --b1=%d",++b,--b1);
  printf("\nb=%d e b1=%d",b,b1);
  printf("\nValores iniciais: j=%d e m=%d",j,m);
  printf("\nm++ - --j vale:%d", m++ - --j);
  printf("\nValores iniciais: k=%d e n=%d",k,n);
  printf("\n++n - k-- vale:%d", ++n - k--);

}
```

- Operadores binários: Atuam sobre dois operandos

Tabela 4.2. Operadores Binários

Operador	*Ação*
+(adição)	Soma dois operandos
- (subtração)	Subtrai o segundo operando do primeiro
*** (multiplicação)**	Multiplica dois operandos
/ (divisão)	Divide o primeiro operando pelo segundo
% (módulo)	Fornece o resto da divisão inteira do primeiro operando pelo segundo

Exemplo:

```
50%4=2          50 |_4__
                 2   12

50/4= 1.25   // se a variável de destino for do tipo float
50/4= 2      // se a variável de destino for do tipo int
50%5=0
```

Tabela 4.3. Precedência entre Operadores Aritméticos

Operadores
++, --
*, /, %
+, -

- Quando dois operandos têm o mesmo nível de precendência, eles são avaliados da esquerda para a direita.

Exemplo:

```
2 + 3 * 4 - 2 * 10 / 2
 2 +  12   -  20  / 2
   14      -    10
           4
```

- Para se alterar a ordem de precedência, devem-se utilizar parênteses.

Exemplo:

```
(2 + 3) * 4 - 2 * (10 / 2)
   5   *  4 - 2 *   5
      20    -     10
           10
```

4.2.1. Conversão de Tipos em Expressões Aritméticas

- As conversões aritméticas implícitas seguem a seguinte regra: *Se um operador binário tiver dois operandos de tipos diferentes, o tipo de "menor" representação é convertido para o de maior representação.*

Exemplo:

```
char ch;
int I;
float f;
double d;
resultado = (ch/i)  +  (f*d)  +  (f+i);
```

int   double   float

int   double   float

double

double

Hierarquia de Conversão

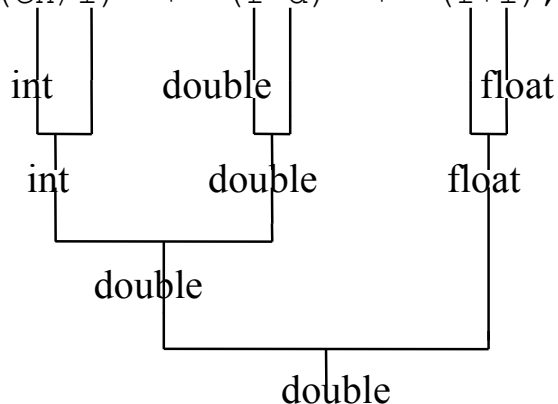
4.3. Operador Molde

- Permite forçar que uma expressão seja de um tipo especificado.

Sintaxe:

```
(tipo) expressão
```

Exemplo:

```
x= 3/2;        // Resulta em 1 porque a parte fracionária é truncada

y= (float) 3/2;        /* Resulta em 1.5 porque 3 é convertido para
                        float antes de se efetuar a divisão*/
```

4.4. Operadores de Atribuição Compostos

- Permitem combinar uma operação matemática binária com uma operação de atribuição de valor.

Sintaxe:

```
expressão_1 operador = expressão_2;
```

é equivalente a:

```
expressão_1 = expressão_1 operador expressão_2;
```

Exemplos:

```
x*=5;    →    x=x*5;
a+=5;    →    a=a+5;
x/=b;    →    x=x/b;
x-=4;    →    x=x-4;
```

- Os operadores de atribuição compostos podem ser usados em operações mais longas e complexas.

Exemplos:

```
m += n+x-y   ⇒   m = m+n+x-y
m /= x*n+y   ⇒   m = m /(x*n+y)
m %= y+m     ⇒   m = m % (y+m)
x += y-= m   ⇒   x = x+(y = y-m)
```

4.5. Operadores Relacionais

São usados para comparar expressões. São avaliados como verdadeiro ($\neq 0$) ou falso (0).

Tabela 4.4. Operadores Relacionais

Operador	*Ação*
`== (igual)`	Verifica se o primeiro operando é igual ao segundo
`> (maior que)`	Verifica se o primeiro operando é maior que o segundo
`< (menor que)`	Verifica se o primeiro operando é menor que o segundo
`>= (maior ou igual a)`	Verifica se o primeiro operando é maior ou igual ao segundo
`<= (menor ou igual a)`	Verifica se o primeiro operando é menor ou igual ao segundo
`!= (não igual a)`	Verifica se o primeiro operando é não igual ao segundo

Exemplos:

```
4 == 3          // avalia como 0 (falso)
3 > 2           // avalia como 1 (verdadeiro)
var= (4 == 3) + (3 > 2);    // atribui 1 a var
```

Tabela 4.5. Precedência dos Operadores Relacionais

Operador
< <= > >=
!= ==

4.6. Operadores Lógicos

Permitem que se usem duas ou mais expressões relacionais em uma única expressão.

Tabela 4.6. Operadores Lógicos

Operador	*Ação*
&& (e)	Avalia como 1 (verdadeiro) se ambas expressões forem verdadeiras, caso contrário, avalia como 0 (falso)
|| (ou)	Avalia como 1 (verdadeiro) se pelo menos uma expressão for verdadeira, caso contrário, avalia como 0 (falso)
! (não)	Avalia como 1 (verdadeiro) se a expressão da direita for falsa e 0 (falso) se a expressão da direita for verdadeira

Exemplos:

```
(5>2)&&(3!=2)     // avalia como 1 (verdadeiro)
  1       1

(3>=2)||(4=2)     // avalia como 1 (verdadeiro)
   1      0

(3<=2)||(4=2)     // avalia como 0 (falso)
   0      0

! (4==2)          // avalia como 1 (verdadeiro)
    0

! (4>3)           // avalia como 0 (falso)
    1
```

Tabela 4.7. Tabela-Verdade dos Operadores Lógicos

p	*q*	*p && q*	*p || q*	*!p*
0	0	0	0	1
0	1	0	1	1
1	1	1	1	0
1	0	0	1	0

Tabela 4.8. Precedência dos Operadores Lógicos

Operador
&&
||

- Expressões conectadas por operadores lógicos são avaliadas da esquerda para a direita e a avaliação pára tão logo a veracidade ou falsidade do resultado for reconhecida.

Em qualquer expressão que espere um valor lógico, o resultado será:

- falso se o valor for 0
- verdadeiro para qualquer outro valor

4.7. Operadores bit a bit

- Permite a manipulação direta com bits individuais de variáveis inteiras.
- Os operadores bit a bit só podem operar sobre variáveis do tipo **int** e **char.**

- Operadores de deslocamento:

Desloca os bits n posições dentro de uma variável inteira

Sintaxe:

Deslocamento à esquerda:

variável << número de posições de bits

Deslocamento à direita:

```
variável >> número de posições de bits
```

- Notar que as operações de deslocamento podem ser usadas para multiplicar e dividir inteiros:

x << y é equivalente a: $x*2^y$

x >> y é equivalente a: $x/2^y$

Exemplos:

```
 x = 00001100        // 00001100 = 12
 x >> 2             // resultado x = 00000011
                   // 00000011 = 3

x = 00001100
x << 2          // resultado x = 00110000
                // 00110000 = 48
```

Tabela 4.9. Operadores lógicos com bits

Operador	*Ação*
~ (complemento de 1)	Inverte o estado de cada bit da variável especificada.
& (e)	Inicializa um bit com valor 1 somente se os bits correspondentes em ambos operandos forem 1, caso contrário, o bit é zerado.
| (ou inclusivo)	Zera um bit somente quando os bits correspondentes em ambos operandos forem 0, caso contrário, o bit é inicializado com o valor 1.
^ (ou exclusivo)	Inicializa um bit com valor 1 somente quando os bits correspondentes em ambos operandos forem diferentes, caso contrário, o bit é zerado.

- A tabela verdade dos operadores **e (&), ou (|)** e **complemento de 1 (~)** é igual à dos seus equivalentes lógicos, exceto por trabalharem bit a bit. O **ou exclusivo (^)** não tem equivalente lógico.

Tabela 4.10. Tabela Verdade para o Ou Exclusivo

p	*q*	*p^q*
0	0	0
1	0	1
1	1	0
0	1	1

Exemplos:

```
 11110000       01010000       11110000
&01010101     |11110000      ^01010101
 01010000      11110000       10100101

     ~(11111101) = (00000010)
```

Exemplo 4.3:

```
/*Desloca.c: Mostra o uso dos operadores de deslocamento*/

#include<stdio.h>
#include<conio.h>

unsigned char y,x=48;
int cont;

main()
{ printf("\nDecimal\t\tDeslocamento p/ esq.\tResultado\n");

 for(cont=1;cont<8;cont++)
 {   y=x<<cont;
     printf("%d\t\t\t%d\t\t%d\n",x,cont,y);
 }
 printf("Pressione qualquer tecla para continuar...\n");
 getch();
 printf("\nDecimal\t\tDeslocamento p/ dir.\tResultado\n");
 for(cont=1;cont<8;cont++)
 {   y=x>>cont;
     printf("%d\t\t\t%d\t\t%d\n",x,cont,y);
 }
 return 0;
}
```

Exemplo 4.4:

```
/* Tstbit.c - Testa os conhecimentos de inglês do usuário e fornece a
   sua nota de acordo com o número de questoes respondidas corretamente.
   Mostra o uso dos operadores lógicos bit a bit */
#include <stdio.h>
#include <conio.h>

char *teste_ingles[]=
{ "1. mother-in-law = mae que anda dentro da lei",
  "2. to dispose = desposar",
  "3. to dispose = dispor",
  "4. busses = plural de onibus",
  "5. bus = beijo estalado",
  "6. bus = onibus",
  "7. naive = ingênuo",
  "8. naive = nave",
  "9. pretty woman = mulher preta",
  "10. to lie = deitar",
  "11. to lie = mentir",
  "12. miss = senhorita",
  "13. to miss = sentir falta",
  "14. to dismiss = rejeitar",
  "15. to dismiss = deixar de ser senhorita",
  "16. misfortune = perder a  fortuna",
};

int obtem_respostas(void);
int conta_bits( int bits_errados );
double testa_nota( int respostas );

#define RESPOSTAS_CORRETAS 0X3C6C

void main()
{     double nota;
      printf("Responda as perguntas com V (verdadeiro) ou F (falso)\n");
      nota = testa_nota( obtem_respostas() );
      printf( "\nSua nota e %3.0f/n", nota);
}

int obtem_respostas()
{     unsigned int respostas = 0;
      int j;
      char c;

      for( j=0; j<=15; j++ ){
            printf("\n%s\n", teste_ingles[j]);
            c = getche();
            if (c == 'v' || c == 'V')
                  respostas |= 1<<j;
                  getch();
      }
      printf("\nRespostas fornecidas=(%x)\n",respostas );
      return respostas;
}

/*    Respostas corretas sao:
        FFVV FVVF FFVV VVFF
*/
```

```
double testa_nota(int respostas)
{     int bits_errados;
      double nota;

      bits_errados = respostas ^ RESPOSTAS_CORRETAS;
      nota=100*((16- conta_bits( bits_errados )) / 16.0);
      return nota;
}

int conta_bits(int bits_errados)
{     int j, conta = 0;

      for( j = 0; j <= 15; j++ )
            if(  bits_errados & (1<<j))
                  ++conta;
      return conta;
}
```

4.8. Operador condicional (? :)

- Opera sobre três operandos, sendo assim um operador ternário.

Sintaxe:

```
expressão_1 ? expressão_2 : expressão_3;
```

se expressão_1 for verdadeira (diferente de 0), atribui como resultado o valor da expressão_2, caso contrário, atribui como resultado o valor da expressão_3.

Exemplo:

```
c = (a==b) ? 4 : 5;    // se a==b então c=4
                       // caso contrário c=5
```

4.9. Operador Vírgula (,)

- Usado para encadear diversas expressões.

- A vírgula provoca uma seqüência de operações. Quando usada do lado direito de uma sentença de atribuição, o valor atribuído é o valor da última expressão da lista separada por vírgulas.

Exemplo:

```
x = (y=3, y+1);   /* Primeiro atribui o valor 3 a y
                     e depois o valor 4 a x */
```

Exemplo 4.5:

```
/* Condic.c - Mostra o uso do operador condicional */
#include <stdio.h>
#include <stdlib.h>

void main()
{ int anos;
  char *mensagem;
  mensagem = (char*)malloc(50);
  printf("\nQuantos anos voce tem?\n");
  scanf("%d",&anos);
  mensagem = (anos>30) ? ( (anos>40) ? "Cuidado, dos enta nao sai mais!" :\
             "Entao voce ja e um(a) balzaquiano(a)!" ) : \
             "Voce ainda esta na flor da idade!" ;
  printf("%s\n",mensagem);
}
```

Exemplo 4.6:

```
/* Virgula.c - Mostra o uso do operador vírgula */
#include <stdio.h>

void main()
{      int i;
       int x,y;
       printf("i\ty\tx\n");
       for(i=0; i<10; i++)
       {      x = ( y=i, y++ );
              printf("%d\t%d\t%d\n",i,y,x);
       }
}
```

Tabela 4.11. Precedência de todos os operadores

Operadores
( ), [ ], ->
!, ~, ++, - -, - (unário), + (unário), *, (tipo), sizeof
***, /, %**
+, -
<<, >>
<, <=, >, >=
= =, !=
&
^
|
&&
||
?:
=, +=, -=, *=, /=, %=, >>=, <<=, &=, ^=
,

Lista de Exercícios Nº 4

1- *Escreva e compile um programa que recebe três números inteiros e usando o operador condicional mostra qual deles é o maior e qual deles é o menor.*

2- *Escreva e compile um programa que recebe um número decimal inteiro e retorna a quantidade de bits iguais a zero no número.*

3- *Escreva e compile um programa que recebe dois inteiros e mostra na tela o resultado da avaliação dos operadores lógicos bit a bit entre estes dois números. Interprete o resultado para um par de números.*

4- *Escreva e compile um programa que lê 10 números e mostra na tela o resultado da soma e da média. Use operadores de atribuição compostos.*

5- *Para x=5 e y=4, avalie o valor das expressões, levando em conta a precedência entre operadores:*

a) *x++ * ++y*
b) *++y - x++*
c) *x++/(y - x)*
d) *x<<2 - y>>1*
e) *(x|y) + (x&y)*
f) *x += y%2*

Agora escreva um programa para verificar se a sua avaliação está correta.

5. Controle de fluxo

Os comandos de controle de fluxo direcionam a execução do programa.

5.1. A instrução `if-else`

- Usada para decidir que comando, ou bloco de comandos será executado, com base no valor de uma expressão.

Sintaxe:

```
if (expressão)
    BLOCO 1;    // Executado apenas se expressão ≠ 0
else
    BLOCO 2;    // Executado apenas se expressão = 0
```

Como funciona:

- O valor de **expressão** é avaliado. Se for verdadeiro, ou seja, se tiver um valor diferente de zero, o BLOCO 1 de comandos é executado, caso contrário, o BLOCO 2 de comandos é executado.
- Se BLOCO1 ou BLOCO2 consistirem de mais de um comando (comando composto), a seqüência de comandos deve ser precedida por uma chave "{" e finalizada com uma chave "}", marcando o início e o fim do bloco.

- Pode-se aninhar comandos if-else.

```
if (expressão1)
   if (expressão2)
       BLOCO 1;
   else
       BLOCO 2;
else
     BLOCO 3;
```

- Deve-se tomar cuidado ao se usar a seguinte sequência:

```
if (expressão1)
   if (expressão2)
       BLOCO 1;
   else
       BLOCO 2;
```

pois o comando **else**, neste caso, se refere ao **if** mais interno.

Para que ele corresponda ao primeiro if, devem-se utilizar chaves:

```
if (expressão1)
{
    if (expressão2)
       BLOCO 1;
}
else
    BLOCO 2;
```

- O comando **else** é opcional.

- A identação auxilia na compreensão do programa, porém o compilador não a leva em consideração.

Exemplo 5.1:

```
/* Avalia.c - Dadas as notas de quatro provas, verifica se o
 aluno foi aprovado ou reprovado. Ilustra o uso de if-else */

#include <stdio.h>
#include <conio.h>
#define NOTA_MIN 6
#define NOTA_MIN_EXAME 4

float nota_1, nota_2, nota_3, nota_4;
float media;

void main(  )
{ printf("\nQual a primeira nota do aluno?");
  scanf("%f",&nota_1);
  printf("\nQual a segunda nota do aluno?");
  scanf("%f",&nota_2);
  printf("\nQual a terceira nota do aluno?");
  scanf("%f",&nota_3);
  printf("\nQual a quarta nota do aluno?");
  scanf("%f",&nota_4);
  media=(nota_1 + nota_2 + nota_3 + nota_4) /4;

  if (media>=NOTA_MIN)
     printf("\nO Aluno foi aprovado com média %.2f",media);
  else  if (media <= NOTA_MIN_EXAME)
          printf("\nO Aluno ficou de exame na disciplina");
        else
          printf("\nO   Aluno   ficou   de   DP   na   disciplina
(reprovado)");

  getch(); // pára a tela
  printf("\n Fim do programa");
}
```

5.2. A instrução `switch`

- Comando de seleção múltipla que testa sucessivamente o valor de uma expressão contra uma lista de constantes inteiras ou de caracteres. Quando o valor coincide, os comandos associados àquela constante são executados.

Sintaxe:

```
switch (expressão)
{   case  constante_1:
          comandos_1;
          break;
    case  constante_2:
          comandos_2;
          break;
                      ...
    case constante_n:
          comandos_n;
          break;
    default:
         comandos_default ;
}
```

Como funciona:

expressão corresponde a um valor inteiro do tipo long, int ou char. Compara-se este resultado com o valor de *constante_1* até *constante_n*.

- Se for encontrada alguma equivalência, o programa passa a executar os comandos a partir deste ponto, até encontrar uma instrução **break**.
- Se nenhuma equivalência for encontrada os comandos relativos a default são executados.
- Caso não haja a opção default, nenhum comando é executado.

Exemplos:

```
switch(n)
{    case 1:
          printf("O valor de n é 1");
          break;
     case 2:
          printf("O valor de n é 2");
          break;
     default:
          printf("O valor de n é diferente de 1 e de 2");
}
```

```
switch (letra)
{     case 'a':
      case 'e':
      case 'i':
      case 'o':
      case 'u':
            printf("\n Você digitou uma vogal");
            break;
      case 'b':
      case 'c':
      case 'd':
            printf("\n Você digitou uma constante válida");
            break;
      default:
            printf("\n Você digitou uma constante inválida");
}
```

Exemplo 5.2:

```
/* Switch.c - Avalia o conhecimento do usuário */

#include <stdio.h>
#include <conio.h>

int opcao;

void main( )
{   printf("\nTeste de avaliação de conhecimento\n");
    printf("\nPergunta: Em que ano o Brasil foi descoberto?");
    printf("\n 1- Em 100 AC por Pitágoras");
    printf("\n 2- Em 1492 por Cristovão Colombo");
    printf("\n 3- Em 1500 por Pero Vaz de Caminha");
    printf("\n 4- Em 1500 por Pedro Alvares Cabral");
    printf("\n Qual a sua escolha? ");
    scanf("%d",&opcao);

    switch(opcao)
    { case 1:
              printf("\n Voce precisa ler mais sobre a história grega !");
              break;
      case 2:
              printf("\n  Chegou  nos  nossos  vizinhos...Mais  sorte  na  próxima
vez!");
              break;
      case 3:
              printf("\n Naturalmente você é um fã da carta do descobrimento !");
              break;
      case 4:
              printf("\n Resposta certíssima !");
              break;
      default: printf("\nVoce não escolheu nenhuma das alternativas válidas.");
    }
    printf("\n Fim do programa ");
    getch();
}
```

5.3. A Instrução `for`

- Permite repetir um bloco de comandos, um determinado número de vezes.

<u>Sintaxe</u>: **`for`** **(**`inicial; condição; incremento`**)**
```
                  bloco_de_comandos;
```

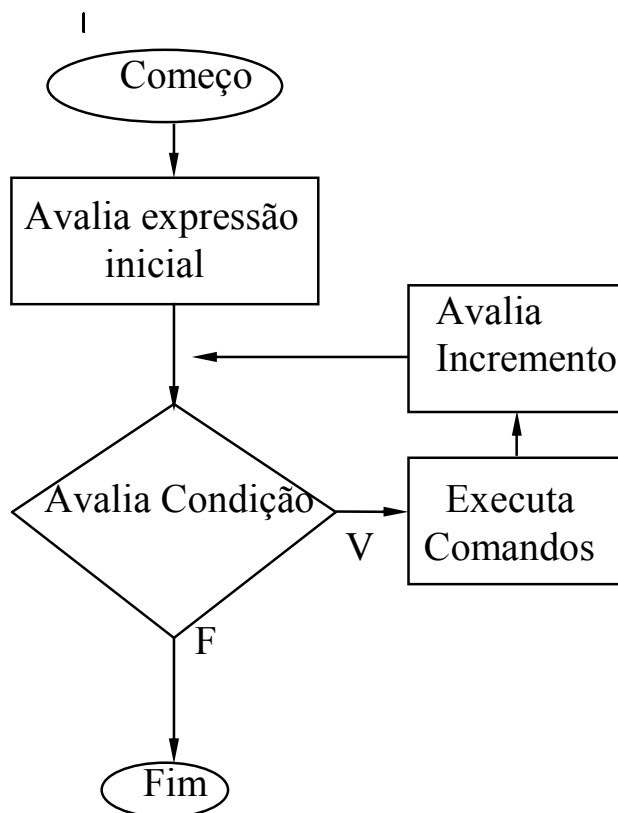
Como funciona:

As expressões *inicial* e *incremento* são, em geral, atribuições e *condição* é uma expressão relacional.

A expressão *inicial* é avaliada, consistindo geralmente na inicialização de uma variável. Em seguida, é avaliada *condição*:

-Se *condição* é verdadeira, avalia-se a expressão *incremento,* executando-se o bloco de comandos e avaliando-se novamente *condição*.

- Se *condição* for falsa, o loop é encerrado.

Representação esquemática de um comando `for`

<u>*Exemplo:*</u>
```
for (i=3; i<5; i++)
    printf("O valor de i e: %d\n", i);
```

- Pode-se omitir a expressão de *inicialização*, se esta já tiver sido avaliada anteriormente no programa, porém o ponto e vírgula deve permanecer.

Exemplo:

```
i =3;
for ( ;i<5;i++)
    printf("O valor de i é: %d\n", i);
```

- A expressão *inicial* pode ser qualquer expressão válida em C.

Exemplo:

```
i =3;
for (printf("comecou") ;i<5; i++)
    printf("O valor de i é: %d\n", i);
```

- Pode-se também omitir a expressão *incremento*, e avaliá-la no corpo da instrução **for**.

Exemplo:

```
i =3;
for (;i <5;)
{ printf("O valor de i e: %d\n", i);
  i ++;
}
```

- Note-se que quando o corpo da instrução **for** consistir de mais de um comando, separados por **(;)** é necessário que estes sejam colocados entre chaves.

- A expressão *condição* pode ser constituída de expressões conectadas por operadores lógicos.

Exemplos:

```
for(i=0; (i<10)&&(vetor[i])!=0;i++)
   printf("\n %d", vetor[i]);

for(i=0; (i<10)&&(vetor[i]);i++)
   printf("\n %d", vetor[i]);
```

- A instrução **for** pode ser usada para se inicializar um vetor.

Exemplo:

```
for(i=0; i <10; vetor[i++]=0);
```

- Usando vírgulas, pode-se criar expressões constituídas de duas sub-expressões, de forma a se realizar duas tarefas. As sub-expressões serão avaliadas da esquerda para a direita e toda a expressão será avaliada como o valor da expressão da direita.

Exemplo:

```
for(i=0, j=10;  i<10;  i++, j--)
    printf("\n%d, %d",  i, j);
```

- A instrução **for** pode ser aninhada, ou seja, executada dentro de outra instrução **for**.

Exemplo:

```
for(cont=0; cont<10; cont++)
{ for(j=0; j<10; j++)
    printf("\n%d", matriz[cont][j]);
 }
```

- Podem-se omitir as três expressões: *inicial*, *condição* e *incremento*:

Exemplo:

```
for( ; ; )
```

tendo-se então um loop infinito, que pode, no entanto, ser interrompido dentro do corpo da instrução **for** por um **break**.

Exemplo 5.3:

```
/* For.c - Mostra o uso de for aninhado */

#include<stdio.h>

void desenha(int fil, int col);

main()
{
  desenha(10, 30);

  return 0;
}

void desenha(int fil, int col)
{   int i;
    for ( ;fil>0 ; fil--)
    { for(i=col; i>0; i--)
          printf("%c",1);
      printf("\n");
    }
}
```

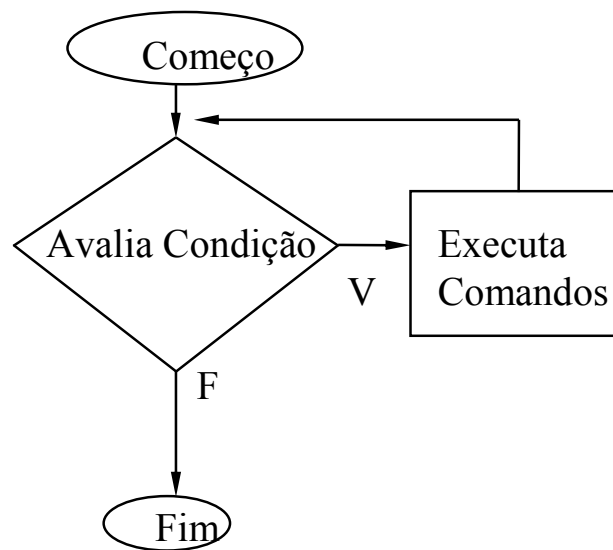
5.4. A instrução **`while`**

- Executa um bloco de instruções enquanto uma determinada condição permanecer verdadeira.

Sintaxe:

```
while (condição)
 comandos;
```

onde *condição* é qualquer expressão válida em C.

Representação esquemática de um comando **`while`**

Exemplo:

```
int i  = -10;
while(i<0)
{   printf("\n%d",i);
    i++;
}
```

- A instrução **`while`** pode ser usada na forma aninhada.

Exemplo 5.4:

```
/* While.c - Solicita quatro números e os retorna na tela
               Ilustra o uso do comando while*/

#include <stdio.h>

int vetor[4];
int indice=0,numero;

main( )
{ printf("Lê quatro números entre 1 e 100 e os mostra na tela \n");
  while (indice<4)
  {   numero=0;
     while( (numero<1) || (numero > 100) )
     {    printf("Digite um número de 1 a 100: ");
          scanf("%d", &numero);
     }
     vetor[indice]=numero;
      indice++;
 }
 printf("\nOs números válidos digitados foram:");
 for (indice=0; indice < 4; indice++)
      printf("\n%d",vetor[indice]);

 return 0;
}
```

5.5. A instrução `do...while`

- Executa um bloco de comandos enquanto uma determinada condição for verdadeira, no entanto, tal condição é avaliada após o bloco ser executado.

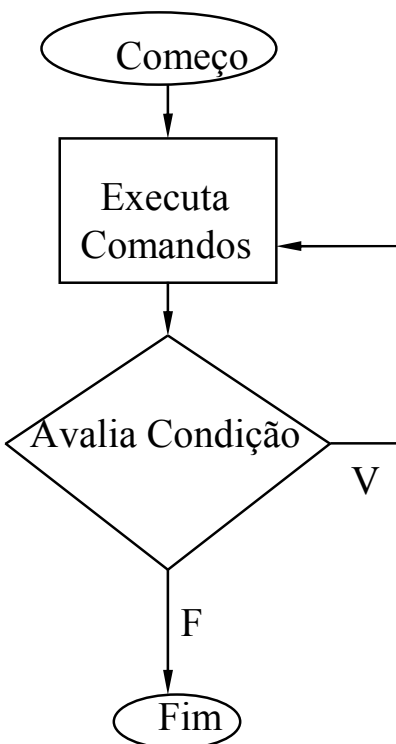
Sintaxe:

```
do
{
    comandos;
} while(condição);
```

sendo *condição* qualquer expressão válida em C.

Exemplo:

```
i=0;
do
{   i++;
    printf("\n%d",i);
} while(i<10);
```

Representação esquemática de um comando `do while`

Exemplo 5.5:

```
/*......................................
   Menu1.c - Operacionaliza um menu simples

   Ilustra o uso do comando "do while "
.........................................*/

#include <stdio.h>
#include <conio.h>
void main(void)
{ int opcao;

  clrscr();
  puts("**********************************************");
  puts(" Monta uma agenda de telefones");
  puts("**********************************************");
  puts("Pressione uma tecla para comecar...");
  getch();

  for(;;)
  { do
       {  clrscr();
          puts("\nEscolha uma das opcoes:\n");
          puts("1) Sair do programa.");
          puts("2) Inserir dados de seus amigos na agenda.");
          puts("3) Ver as informacoes contidas na sua agenda.");
          puts("4) Obter o telefone de um amigo.");
          printf("\nSua opcao: ");
          scanf("%d",&opcao);
          fflush(stdin);
        } while(opcao<1 || opcao>4);
```

```
        switch(opcao)
        {    case 1:
                   printf("\nFim do programa");
                   getch();
                   exit(0); // interrompe o for (;;)
                   break;

             case 2:
                   printf ("\nInsere dados dos amigos");
                   getch();
                   break;

             case 3:
                   printf ("\nImprime dados da agenda");
                   getch();
                   break;

             case 4:
                   printf ("\nConsulta dados dos amigos");
                   getch();
                   break;

             default:
                   puts("Voce nao escolheu uma opcao valida!");
        }
    }
}
```

5.6. Comandos de Desvios

5.6.1. O comando de desvio `break`

- Pode ser usado para terminar um **`case`** em um comando **`switch`** ou para forçar uma terminação imediata de um loop.

Sintaxe:

```
break;
```

Exemplo:

```
int t;
for( t=0; t<100; t++ )
 {   printf("%d ", t);
     if( t==10 )
          break;
 }
```

- Um comando **`break`** provoca uma saída apenas do loop mais interno.

Exemplo 5.6:

```
/* Break.c: Mostra todos os numeros divisíveis por 2 entre 0 e 20
           Ilustra o uso do comando break         */
#include <stdio.h>

void main()
{     int i,j;

      printf("Numeros divisíveis por 2 entre 0 e 20\n");

      i = 0;
      for (; i<=20; )
       {
            for( ; i<=20;i++)
             {  if (i%2)
                { i++;
                  break;
                }
                printf("%d\n",i);
            }
      }
}
```

5.6.2. O comando de desvio `continue`

- Permite que se comece a próxima iteração de um loop imediatamente.

<u>Sintaxe</u>:

```
continue;
```

<u>Exemplo:</u>

```
for (i=0; i<n; i++)
 { if(a[i]<0)
        continue;
   a[i]++; /* Se a[i]<0 esta instruçãonão é executada! */
 }
```

Exemplo 5.7:

```
/* Continue.c - Exibe todos os numeros divisíveis por 3
                entre 0 e 30.
                Ilustra o uso do comando continue */

#include <stdio.h>

void main()
{    int i;

     printf("Numeros divisíveis por 3 entre 0 e 30\n");
     for( i=0; i<=30; i++)
     {
          if (i%3)
               continue;
          printf("%d\n",i);
     }
}
```

5.6.3. O comando de desvio `goto`

- Permite saltos ou desvios incondicionais.

Sintaxe:

```
goto destino;
```

sendo *destino*, um rótulo, ou um label, que identifica o ponto para onde o programa deve prosseguir.

Exemplo:

```
if(n==1)
     goto ponto1;
else  if(n= = 2)
          goto ponto2;
ponto1:
    printf("\nO valor de n é 1");
    goto fim;
ponto2:
    printf("\nO valor de n é 2");
    goto fim;
fim:;
```

Exemplo 5.8:

```
/* Goto.c: Ilustra o uso do comando goto  na construção
   de um loop */

#include <stdio.h>

void main()
{    int x = 0;

     loop1:
          x++;
          if(x<=10)
          {    printf("x = %d\n", x);
               goto loop1;
          }
}
```

Deve-se evitar o uso do `goto`, pois sempre é possível usar uma outra instrução que realize a mesma tarefa que o `goto` e torne o programa mais fácil de ser entendido.

5.6.4. A função `exit()`

- Da mesma forma que se pode sair de um laço, com a instrução **`break`**, pode-se sair do programa com a função **`exit()`** da biblioteca padrão.

- Sintaxe:

```
exit(int código_de_retorno);
```

- O valor de código de retorno é retornado ao processo chamador que é, normalmente, o sistema operacional. O zero é usado, em geral, como um código de retorno que indica uma terminação normal do programa.

Exemplo:

```
int num;
printf("Forneça um número inteiro positivo\");
scanf("%d",&num);
if (num<0)
{  printf("O número deveria ser positivo!\n");
   exit(0);
}
```

Lista de Exercícios Nº 5

1- *Altere o programa do exemplo 5.1 para tirar a média de 6 provas. Mude a nota mínima de aprovação para 7.*

2- *Altere o programa do exemplo 5.1 para possibilitar a seguinte classificação:*

$7 \leq$ *média* ≤ 10 *- aprovado sem exame*
$6 \leq$ *média* < 7 *- aprovado se entregar um trabalho*
$3 \leq$ *média* < 6 *- exame*
$0 \leq$ *média* < 3 *- reprovado sem direito a exame*

3- *Reescreva o programa do exemplo 5.3, para que, ao invés de um quadrado, seja desenhado um triângulo.*

4- *Use o loop do...while no programa do exemplo 5.1 para evitar que seja fornecida uma nota menor que zero e maior que dez.*

5- *Use a instrução switch para implementar um programa de voto eletrônico. Considere uma eleição com 5 candidatos: A, B, C, D e E.*

6- *Reescreva o programa do exemplo 5.8 utilizando um loop for.*

7. *Reescreva o programa 5.3 usando while ao invés de for.*

6- Funções

- Realizam tarefas específicas.
- Recebem informações através de argumentos.
- São designadas por um nome e podem retornar um valor à função que as chamou.
- É necessário que o programa contenha o **protótipo** e a **definição** da função.

Vantagens em se utilizar funções:
- Possibilitam a realização de uma mesma tarefa a partir de vários pontos do programa, sem a necessidade de se repetir código.
- Permitem que se isole o código e as variáveis do restante do programa.
- Tornam mais fácil a depuração do programa.

6.1. O protótipo de uma função

- Fornece ao compilador uma descrição da função que será definida posteriormente.
- Deve ser colocado antes da primeira chamada à função, de preferência antes da função "main".
- Deve possuir o tipo de variável de retorno, nome da função, os tipos de variáveis que são os parâmetros formais e, opcionalmente, nomes associados a cada um destes tipos.

Exemplos:

```
float area(float lado, float altura);
int mmc(int, int);
```

6.2. Definição da função

- A primeira linha de definição de uma função, chamada de cabeçalho da função, deve ser idêntica ao protótipo da função, com exceção de:
    - não deve haver o ponto e vírgula final;
    - o nome dos parâmetros formais passa a ser obrigatório.
- Deve conter as instruções necessárias para que a tarefa desejada seja realizada, devendo estas instruções estarem contidas entre um abre e fecha chaves "{}", que é chamado de corpo da função.

Exemplo:

```
/* calcula a área de um triângulo equilátero */
float area(float lado, float altura)
{ float result;
  result = (lado*lado)*sqrt(3)/4;
  return(result);
}
```

6.3. O tipo de retorno de uma função

- Especifica o tipo de dado que a função deverá retornar.
- Pode ser qualquer dos tipos de dados válidos em C.
- Se nenhum resultado for retornado, tanto o protótipo quanto a definição devem ter como tipo de retorno a designação **void**.
- Se não for declarada como do tipo **void**, deve-se incluir em algum ponto do corpo da função a instrução **return** para que seja retornado um valor à função que a chamou.
- Se o tipo de retorno for omitido considera-se o mesmo como sendo do tipo **int**, por *default*.

6.4. O nome de uma função

- Pode conter letras, números e o caracter de sublinhado.
- O primeiro caracter deve ser uma letra ou caracter de sublinhado.
- Letras maiúsculas e minúsculas correspondem a caracteres diferentes.
- Não podem ser usadas palavras reservadas.

6.5. A lista de parâmetros formais

- Deve conter um item correspondendo a cada argumento, especificando o tipo de dado e o nome do parâmetro.

Exemplos:

```
long soma(int a, int b)
float volume(int a, int b, int h)
```

- Quando nenhum argumento é passado à função ou nenhum argumento é retornado pela função, deve-se usar a palavra **void**.

Exemplos:

```
void listas(int x)
float aleatorio(void)
void menu(int x)
void mensagem (void)
```

6.6. Instruções dentro de funções

- Podem ser utilizadas todas as instruções do C.
- Não podem ser definidas funções dentro de outras funções.
- Podem chamar outras funções.

6.7. Retorno de uma função

- Deve-se usar a palavra **return** seguida de uma expressão válida em C, que após ser avaliada, tem o seu valor retornado.

Exemplos:

```
return x;
return (a*b);
```

- O corpo da função pode ter várias instruções **return**, porém só a expressão de uma delas é avaliada.

Exemplo:

```
if(a>b)
   return a;
else
   return b;
```

6.8. Chamada de uma função

- Uma função é chamada mencionando-se o seu nome e a lista de argumentos colocada entre parênteses.

Exemplos:

```
volume(x,y,z);

total=volume(x,y,z);

printf("\n%d", volume(x,y,z));

if (volume(x,y,z) > 10)
    printf("\nEsta é uma caixa grande");
```

Exemplo 6.1:

```
/* Potencia.c: utiliza uma funcao que eleva "x" a "y" */

#include <stdio.h>
#include <conio.h>
#include <math.h>
int base, expoente;
float result;
float potencia (int bas, int exp);

void main(void)
{clrscr();
 printf("\nPrograma que calcula o valor de x elevado a y");
 printf("\nDigite o valor da base (inteiro): ");
 scanf("%d", &base);

 while (base != 0)
 {  printf("\nDigite o valor do expoente (inteiro): ");
    scanf("%d", &expoente);
    result = potencia(base,expoente);
    printf("\n %d elevado a %d = %.4f ",base,expoente,result);
    printf("\nDigite o valor da base (inteiro) 0 para sair: ");
    scanf("%d", &base);
 }
 printf("\nFim do programa");
 getch();
}

float potencia(int bas, int exp)
{ float result=1;
  int i;

  if (exp == 0)
     return 1;
  if (exp > 0)
  { for (i=0; i<exp; i++)
       result *= base;
  }
  else /* expoente é negativo, inverte a base */
  { for (i=0; i<abs(exp); i++)
       result *= (float)(1.0/base);
  }
  return result;
}
```

Exemplo 6.2:

```
/* funcao1.c - Avalia o conhecimento do usuário */
#include <stdio.h>
#include <conio.h>

int escolha;
void menu(void);
void avaliacao(int x);

void main( )
{  clrscr();
   printf("\nTeste de avaliacao de conhecimento: \n");
   menu( );
   getch();
}

void menu(void)
{ int opcao;

  printf("\nPergunta: Em que ano o Brasil foi descoberto?");
  printf("\n1- Em 100 AC por Pitagoras");
  printf("\n2- Em 1100 por um vicking entediado chamado Kulm");
  printf("\n3- Em 1990 por Chico Buarque de Holanda");
  printf("\n4- Em 1500 por Pedro Alvares Cabral");
  printf("\nQual a sua escolha?  ");
  scanf("%d",&opcao);
  avaliacao(opcao);
}

void avaliacao(int x)
{   switch(x)
    {  case 1:
         printf("\n Voce precisa ler mais sobre a historia grega!");
         break;
        case 2:
              printf("\n Mais sorte na proxima vez!");
              break;
        case 3:
              printf("\n Naturalmente voce eh um fa do cantor!");
              break;
        case 4:
              printf("\n Resposta certissima !");
              break;
        default:
          printf("\nVoce nao escolheu nenhuma das alternativas validas.");
    }
}
```

Exemplo 6.3:

```
/* funcao2.c - Operacionaliza um menu simples */
#include <stdio.h>
#include <conio.h>

int selecionar_menu(void);
int opcao;
void main( )
{    opcao = selecionar_menu( );
     printf("Voce escolheu a opcao %d", opcao);
     getch();
}

int selecionar_menu(void)
{ int selecao=0;
      do
      { printf("\n");
        printf("\n1 - Acrescentar um registro");
        printf("\n2 - Alterar um registro");
        printf("\n3 - Apagar um registro");
        printf("\n4 - Sair");
        printf("\n");
        printf("\nDigite a escolha:");
        scanf("%d", &selecao);
       }while(selecao<1 || selecao >4);
       return selecao;
}
```

Exemplo 6.4:

```
/* funcao3.c - Dadas as notas de quatro provas, verifica se o
               aluno foi aprovado ou reprovado */
#include <stdio.h>
#define NOTA_MIN 7
float nota_1, nota_2, nota_3, nota_4;
float media;

float calcula_media(float x1, float x2, float x3, float x4);

void main(  )
{ printf("\nQual a primeira nota do aluno?");
  scanf("%f",&nota_1);
  printf("\nQual a segunda nota do aluno?");
  scanf("%f",&nota_2);
  printf("\nQual a terceira nota do aluno?");
  scanf("%f",&nota_3);
  printf("\nQual a quarta nota do aluno?");
  scanf("%f",&nota_4);
  media=calcula_media(nota_1, nota_2, nota_3, nota_4);
  if(media>=NOTA_MIN)
     printf("\nO Aluno foi aprovado com média %.2f",media);
  else
     printf("\nO Aluno foi reprovado. Nota insuficiente: %.2f",media);
  getch();
}

float calcula_media(float x1, float x2, float x3, float x4)
{  float soma, media;
   soma=x1+x2+x3+x4;
   media = soma/4;
   return(media);
}
```

6.9. Recursão

- Uma função em C pode chamar a si própria.

Exemplo 6.5:

```
/* Fatorial.c: calcula o fatorial de um número recursivamente */

#include <stdio.h>

double resultado;
int numero;
double fatorial(int valor);

void main(void)
{
 printf("\nPrograma que calcula o fatorial de um numero.");
 printf("\nDigite o numero: ");
 scanf("%d", &numero);

 resultado=fatorial(numero);
 printf("\nO fatorial de %d = %g", numero, resultado);
 printf("\nFim do programa");
 getch();
}

double fatorial(int valor)
{ if(valor==1)
      return 1;
 else
     {  valor*=fatorial(valor-1);
        return valor;
     }
}
```

6.10. Regras de escopo

- Escopo de uma variável (ou constante) é o termo técnico que denota a região do programa fonte C onde a variável ou constante é reconhecida ou está ativa.

- A duração de uma variável se refere ao tempo de vida de armazenamento desta na memória principal.

6.10.1. Variáveis Locais

- Declaradas dentro de um bloco de código.

Bloco de Código: Porção de código delimitada por um abre-chave “{” e um fecha-chave “}” .

- Só podem ser referenciadas dentro do bloco de código onde tenham sido declaradas, não sendo reconhecidas fora dele.

- Devem ser declaradas no início do bloco de código, antes de qualquer comando.

- Existem apenas enquanto o seu bloco de código está sendo executado: são criadas na entrada e destruídas na saída, não retendo, portanto, seus valores entre chamadas.

- Podem receber um endereço diferente a cada vez que o bloco é executado.

Exemplos:

```
float func1(void)
{  float x;
   x = 10;
    return (sqrt(x));
}

float func2(void)
{  float x;
   x = 200.10;
    return(x);
}

/* O x em func1 não tem absolutamente qualquer relação com
   o x de func2, mas ambos são variáveis locais */


for (i=0; i<10; i++)
 {   int j = 2;
     printf("%d\n",j*i);
  }

/*  j só é visível dentro do corpo da instrução for */
```

- Os parâmetros formais de uma função se comportam como variáveis locais.

Exemplo 6.6:

```
/* Locais.c - Mostra o comportamento das variáveis locais */
#include <stdio.h>

void local1(void);
void local2(int x, int y);

void main(void)
{ int var1=1, var2=2;

   printf("\nValor das variaveis antes de chamar a funcao:
            var1=%d, var2=%d",var1,var2);
   local1();
   printf("\nValor das variaveis depois de chamar a funcao
           local1(): var1=%d, var2=%d",var1,var2);

   local2(var1, var2);
   printf("\nValor das variaveis depois de chamar a funcao
           local2(): var1=%d, var2=%d",var1,var2);

   printf("\nFim do programa");
}

void local1(void)
{
   int var1=100, var2=100;

   printf("\nValor das variaveis dentro da funcao local1():
            var1=%d, var2=%d",var1,var2);
}

void local2(int var1, int var2)
{
   var1 += 100;
   var2 += 200;
   printf("\nValor das variaveis dentro da funcao local2():
            var1=%d, var2=%d",var1,var2);
}
```

6.10.2. Variáveis Globais

- São declaradas fora de qualquer função, em qualquer lugar, anterior à primeira vez em que são usadas.
- Podem ser usadas em qualquer bloco de código.
- Retém o seu valor durante toda a execução do programa.

Obs: Se uma variável local é declarada em um bloco de código com o mesmo nome que uma variável global, qualquer referência a este nome dentro do bloco de código em que foi declarada diz respeito à variável local e não à global.

- Variáveis globais só devem ser usadas quando se necessita de um mesmo dado em muitas funções do programa, pois o seu uso tem pontos negativos:

- Ocupam memória durante todo o tempo de execução do programa;
- Podem gerar a ocorrência de erros devido a efeitos colaterais;
- Tornam mais difícil a depuração do programa.

Exemplo 6.7:

```
/* Globais.c - Mostra como se comportam as variáveis globais */

#include <stdio.h>

int var1=1, var2=2;

void funcao1(void);
void funcao2(void);

void main(void)
{

   printf("\nValor das variaveis antes de chamar a funcao1():
             var1=%d, var2=%d",var1,var2);
   funcao1();
   printf("\nValor das variaveis depois de chamar a funcao1():
          var1=%d, var2=%d",var1,var2);
   funcao2();
   printf("\nValor das variaveis depois de chamar a funcao2():
          var1=%d, var2=%d",var1,var2);

   printf("\nFim do programa");
}

void funcao1(void)
{
    int var1=100, var2=100;

    printf("\nValor das variaveis dentro da funcao1():
           var1=%d, var2=%d",var1,var2);
}

void funcao2(void)
{
   var1 += 1;
   var2 += 2;
   printf("\nValor das variaveis dentro da funcao2():
           var1=%d, var2=%d",var1,var2);
}
```

6.10.3. A declaração `extern`

- Um programa extenso pode se encontrar distribuído em vários arquivos, que são compilados separadamente e "linkeditados" juntos.

- Uma variável global pode ser usada em funções que se encontrem em diferentes arquivos, mas só pode ser declarada uma única vez.

- Para informar a todos os arquivos do programa a respeito da existência de uma variável global, deve-se declará-la em um deles e nos outros deve-se preceder à sua declaração com a palavra-chave **extern**, o que faz com que deixe de ser uma declaração para ser uma referência.

- Se se deseja atribuir um valor à variável quando da sua declaração, esta atribuição só deve ser feita uma vez no arquivo em que é declarada.

Exemplo 6.8:

```
/* Extern1.c: Juntamente com volume.c compõem o programa de cálculo de
   volumes que ilustra o uso de funções declaradas como extern */

#include<stdio.h>
#include<stdlib.h>

extern void volume(int opcao);

float raio, altura;

void main(void)
{ int opcao;
  puts("Calculando volumes.\n");

  for(;;)
  {   puts("\nEscolha uma das opcoes que seguem:");
      puts("1) Volume da esfera.");
      puts("2) Volume do cubo.");
      puts("3) Volume do cilindro.");
      puts("4) Sair do programa.");
      fscanf(stdin,"%d", &opcao);
      if(opcao < 1 || opcao > 4)
        puts("\nA opcao escolhida nao e valida");
      else
        if (opcao == 4)
           exit(0);
        else
           volume(opcao);
  }
}
```

Exemplo 6.9:

```
/* volume.c */
#include <stdio.h>
#define ESFERA   1
#define CUBO     2
#define CILINDRO 3

extern float raio, altura;

void volume(int opcao)
{ switch(opcao)
 {      case ESFERA:
          {    #define PI 3.1416
               float volume;
               puts("Forneca o raio da esfera:");
               fscanf(stdin,"%f",&raio);
               volume = (float)3/4*PI*raio*raio*raio;
               printf("\n O volume da esfera e %f:\n",volume);
           }
          break;
        case CUBO:
          {    float volume;
               puts("Forneca o lado do cubo:");
               fscanf(stdin,"%f",&raio);
               volume = raio*raio*raio;
               printf("\n O volume do cubo e %f:\n",volume);
           }
          break;
        case CILINDRO:
          {    #define PI 3.1416
               float volume;
               puts("Forneca o raio e a altura do cilindro:");
               fscanf(stdin,"%f %f",&raio, &altura);
               volume = 2*PI*raio*raio*altura;
               printf("\n O volume do cilindro e %f:\n",volume);
           }
          break;
        }
}
```

6.10.4. Variáveis `register`

- Registradores são áreas de armazenamento da CPU e são usados para processar cálculos aritméticos.

- A palavra-chave **register** antecedendo a declaração de uma variável sugere ao compilador o armazenamento de uma variável em um registrador, o que torna mais rápida a execução de qualquer operação envolvendo aquela variável (útil para o caso de variáveis que são acessadas com muita freqüência). Esta sugestão pode ou não ser acatada pelo compilador.

- Não há limite para o número de variáveis que podem ser declaradas como **register.**

- Não se pode fazer referência ao endereço de uma variável **register**, visto que um registrador não é endereçável.

- O especificador **register** só pode ser aplicado à variáveis locais.

Exemplo:

```
register int i;
for(i=0; i<=10; i++)
    printf("i=%d\n",i);
```

6.11. Main como função que recebe argumentos

- Passam-se informações para a função **main** através dos argumentos da linha de comando.

Exemplo:

```
prog     18          1.77
  ↑       ↑            ↑
nome    argumento   argumento
do
programa
```

- Há dois argumentos internos especiais aos quais dá-se tradicionalmente o nome de **argc** e **argv**:

- **argc**:
  - ♦ É um inteiro, declarado como **int argc**;
  - ♦ Contém o número de argumentos da linha de comando;
  - ♦ O nome do programa é interpretado como primeiro argumento, portanto **argc** é pelo menos 1.
- **argv:**
  - ♦ É um ponteiro para uma matriz de ponteiros para caracter;
  - ♦ Cada elemento da matriz aponta para um argumento da linha de comandos;
  - ♦ É declarado como **char *argv[]**, onde os colchetes vazios indicam que é uma matriz de tamanho a ser determinado.

- Todos os argumentos da linha de comando são interpretados como strings.

Exemplo:

```
main(int argc, char *argv[])

/* Definindo a função main desta forma indica que se
 pretende entrar com argumentos pela linha de comando */
```

Exemplo 6.10:

```
/* Main.c: Ilustra a entrada de argumentos pela linha
           de comando */
#include<stdio.h>
#include<stdlib.h>

void main(int argc, char *argv[])
{int arg;
 if(argc<2)
 {    puts("Forneca algum argumento na linha de comando\n");
      exit(0);
 }

 puts("Voce forneceu os seguintes argumentos:\n");
 for( arg=1; arg<=argc; arg++)
      printf("argv[%d]=%s\n",arg-1,argv[arg-1]);
}
```

6.12. Uso de arquivos cabeçalho (.h)

- É hábito manter em um arquivo separado:
  - Declarações de protótipos de funções;
  - Declarações de variáveis globais;
  - Definições de macros.

- Os arquivos cabeçalho definidos pelo programador têm por convenção a extensão **.h**, da mesma forma que os arquivos cabeçalho que acompanham a biblioteca C.

- Devem ser incluídos no início do arquivo com o código fonte.

<u>Sintaxe</u>:

```
#include "arquivo.h"
```

A procura pelo aqruivo começa normalmente onde o programa fonte se encontra.

ou

```
#include <arquivo.h>
```

A procura pelo arquivo segue uma regra definida pela implementação.

Exemplo 6.11:

```
/* funcao4.c: Ilustra o uso de estruturas,vetores de  estruturas
               e inclusao de arquivos .h  */

#include "funcao4.h"

void main(void)
{ int opcao;
  int numero_de_amigos;

  clrscr();
  puts("*********************************************");
  printf("Seu nome e %s.\n",inf_sobre_voce.nome);
  printf("Seu telefone e %s,\n", inf_sobre_voce.fone);
  printf("A data em que nasceu foi %d/%d/%d.\n",
         inf_sobre_voce.data_nascimento.dia, inf_sobre_voce.data_nascimento.mes,
         inf_sobre_voce.data_nascimento.ano);
  puts("*********************************************");
  getch();

  for(;;)
  { do
      {  clrscr();
       puts("\nEscolha uma das opcoes:\n");
       puts("1) Sair do programa.");
       puts("2) Corrigir informacoes sobre voce.");
       puts("3) Montar uma pequena lista com informacoes sobre seus amigos.");
       puts("4) Ver as informacoes contidas na sua lista.");
       printf("\nSua opcao: ");
       scanf("%d",&opcao);
       fflush(stdin);
     } while(opcao<1 || opcao>4);

      switch(opcao)
       {      case SAIR:
                   printf("\nFim do programa");
                   getch();
                   exit(0);
                   break;

             case CORRIGIR:
                   corrigeInformacoes();
                   getch();
                   break;
             case MONTALISTA:
                   numero_de_amigos = montaLista();
                   getch();
                   break;
             case VERINFORM:
                   verInform(numero_de_amigos);
                   getch();
                   break;
             default:
                   puts("Voce nao escolheu uma opcao valida!");
           }
      }
   }
```

```
void corrigeInformacoes(void)
{
puts("Forneca o seu nome:");
gets(inf_sobre_voce.nome);
puts("Forneca seu telefone:");
gets(inf_sobre_voce.fone);
puts("Forneca o dia do seu nascimento:");
scanf("%d",&inf_sobre_voce.data_nascimento.dia);
puts("Forneca o mes do seu nascimento:");
scanf("%d",&inf_sobre_voce.data_nascimento.mes);
puts("Forneca o ano do seu nascimento:");
scanf("%d",&inf_sobre_voce.data_nascimento.ano);
fflush(stdin);
puts("**********************************************");
printf("Seu nome e %s.\n",inf_sobre_voce.nome);
printf("Seu telefone e %s,\n",inf_sobre_voce.fone);
printf("A data em que nasceu foi %d/%d/%d.\n",
        inf_sobre_voce.data_nascimento.dia,
        inf_sobre_voce.data_nascimento.mes,inf_sobre_voce.data_nascimento.ano);
puts("**********************************************");
}

int montaLista(void)
{   static int conta_amigo = 0;
    char resposta;
    do{ puts("Deseja entrar com informacoes sobre algum amigo?(s/n)");
        resposta = getche();
        if(resposta=='s' || resposta=='S')
        {    puts("\nForneca o nome do seu amigo:");
           gets(agenda[conta_amigo].nome);
           puts("Forneca o telefone do seu amigo:");
           gets(agenda[conta_amigo].fone);
           puts("Forneca o dia do nascimento do seu amigo:");
           scanf("%d",&agenda[conta_amigo].data_nascimento.dia);
           puts("Forneca o mes de nascimento do seu amigo:");
           scanf("%d",&agenda[conta_amigo].data_nascimento.mes);
           puts("Forneca o ano do nascimento do seu amigo:");
           scanf("%d",&agenda[conta_amigo].data_nascimento.ano);
           conta_amigo++;
         }
    fflush(stdin);
    }while(resposta=='s' || resposta=='S');
    return(conta_amigo);
}

void verInform(int numero_de_amigos)
{   int i;
    if (numero_de_amigos == 0)
       printf("Sua lista de amigos esta vazia....");
    else
    {  puts("\nInformacoes sobre seus amigos:\n");
       for(i=0; i<numero_de_amigos; i++)
       {  printf("\nNome: %s\n", agenda[i].nome);
          printf("Telefone: %s\n", agenda[i].fone);
          printf("Data de nascimento: %d/%d/%d\n",
                  agenda[i].data_nascimento.dia,agenda[i].data_nascimento.mes,
                  agenda[i].data_nascimento.ano);
       }
    }

}
```

```
/* funcao4.h: arquivo cabecalho de funcao4.c */

#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

#define SAIR                  1
#define CORRIGIR              2
#define MONTALISTA            3
#define VERINFORM                     4

void corrigeInformacoes(void);
int montaLista(void);
void verInform(int);

struct informacoes
       {   char nome[50];
             char fone[20];
             struct{int dia;
                  int mes;
                         int ano;
                      } data_nascimento;
       }inf_sobre_voce={"Clotilde","3855-6535",11,2,2000};
struct informacoes agenda[20];
```

Lista de Exercícios Nº 6:

1. Escreva um programa que, fornecidos 3 números pelo usuário, calcule o produto do primeiro pela soma dos outros dois. Use para tanto duas funções: uma que retorna a soma de dois números e outra que retorna o produto de dois números.

*2. Escreva um programa que leia vários valores inteiros (**n**) fornecidos pelo usuário (flag=-1) e calcule a soma dos inteiros de 1 a n para cada n lido. Use uma função para realizar o cálculo.*

3. Faça um programa que leia vários pares de números; o primeiro corresponde ao NUMERADOR e o segundo ao DENOMINADOR (flag= NUMERADOR= 0). Em seguida chame duas funções que recebam o numerador e o denominador como parâmetros e devolvam, a primeira o resultado da divisão real entre eles e a segunda, o quociente e o resto da divisão inteira.

4. Faça um programa que leia um arquivo contendo o nome de várias empresas ou pessoas e o seu telefone e armazene os dados num vetor. Em seguida, peça ao usuário o nome das empresas/pessoas (flag="fim") e chame uma função que receba o nome como parâmetro e devolva o número do telefone correspondente, telefone este que deve ser informado ao usuário.

*5. Escreva um programa que leia vários valores reais (**n**) fornecidos pelo usuário (flag=0) e chame uma função que calcule a sua raiz quadrada e informe se a raiz é exata ou não.*

6. Escreva um programa que leia a data de nascimento de várias pessoas (dia, mês e ano) (flag= dia= -1) e chame uma função que calcule e imprima a sua idade.

*7. Escreva um programa que leia vários valores inteiros (**n**) fornecidos pelo usuário (flag=0) e chame uma função que verifique se o número é primo ou não. (Sugestão: tente dividir o número po 2, 3, 4, .. n; se alguma destas divisões resultar em zero ele não é primo...).*

8. Faça um programa que forneça um menu de opções para o usuário: calcular a área de 1- quadrado, 2-círculo, 3- elipse, 4-triângulo equilátero e, de acordo com a opção escolhida, chamar a função que faça o cálculo desejado pedindo os parâmetros adequados. (A(quadrado)= a^2; A(círculo)= πr^2; A(elipse)=πab; A(triang. equil)= $a^2\sqrt{3} / 4$).

9. Escreva um programa que receba três ou mais palavras como argumentos de linha de comando e as imprima na tela na ordem inversa em que foram fornecidas.

10. Escreva um programa que leia vários valores de temperaturas em graus Fahreinheit fornecidos pelo usuário (flag=1000) e chame uma função que calcule e imprima a temperatura em graus Celsius. (Celsius= (Fahreinheit.-32) + (5/9)).

7. Vetores e Matrizes

- Um vetor ou uma matriz é uma coleção de variáveis do mesmo tipo armazenadas contiguamente na memória.
- Um vetor ou uma matriz pode ser declarada como sendo de qualquer um dos tipos primitivos do C.

7.1. Vetores (matrizes unidimensionais)

- Declaração de um vetor:

Sintaxe:

```
tipo nome[tamanho];
```

Exemplo:

```
int idades[10]; /* Vetor chamado idades com 10 elementos */
```

- Referência a um elemento de um vetor:

Uma referência a um elemento de um vetor é feita com o uso de índices, que indicam a posição do elemento dentro do vetor.

Exemplos:

```
id = idades[0];
   /* Atribui a id o primeiro elemento do vetor idades*/
idades[1]=idades[2]+idades[3];
  /* Atribui ao segundo elemento a soma do terceiro com
   o quarto elemento do vetor idades */
```

Obs: Todo vetor tem 0 (zero) como índice do seu primeiro elemento.

- Inicialização de vetores:

- Podem ser inicializados quando da sua declaração.

  Exemplos:

```
int idades[10] = {1,2,3,4,5,6,7,8,9,10};
     /* Inicializa todos os elementos */

int idades[10] = {1,2,3,4,5};
   /*Inicializa os 5 primeiros elementos*/
```

- Podem ser inicializadas após serem declaradas.

  Exemplos:

```
idades[0] = 1;
idades[1] = 2;

for(i=2; i<10; i++)
   idades[i] = i;

for(i=0; i<10; idades[i++]=0);
```

- **Exercício 7.1:**

Faça um programa para ler a nota de 80 alunos e imprimir a nota e a média da turma para cada aluno.

➔ Se a turma tivesse 3 alunos, e não soubéssemos utilizar os vetores, seriam necessárias 5 variáveis para a solução:

```
float n1, n2, n3, total, media;
void main(void)
{      ...
            scanf("%f %f %f ",&n1,&n2,&n3);
            total = n1 + n2 + n3;
            media = total/3;
            printf("Nota=%f Media=%f",n1, media);
       ...
}
```

➔ Para a turma de 80 alunos, seriam necessárias 82 variáveis!!
➔E se fossem os candidatos ao vestibular (4000 alunos)?

➔ Esse é um típico problema em que a solução nos obriga a utilizar vetores:

```
float notas[80];
```

➔ Como fazer atribuição das notas no vetor?

```
notas[0] = 5.0;
notas[3] = 7.2;
notas[2] = 3.0;
```

➔ Como ler todas as notas do teclado?

```
for (i=0; i<80; i++)
 { printf("\nDigite a nota do aluno %i: ",i);
   scanf("%f",&notas[i]);
 }
```

Exemplo 7.1:

```
/*vetor1.c: calcula a média das notas de uma turma*/
#include <stdio.h>
#include <conio.h>
#define MAX  80

float notas[MAX], total=0, media;
int i;

void main(void)
{ clrscr();
  printf("\nEste programa calcula a media de notas da turma");

  for (i=0; i<MAX; i++)
  { printf("\nDigite a nota do aluno %d:",i);
    scanf("%f",&notas[i]);
    total += notas[i];
  }

  media = (float)total / MAX;
  for (i=0; i < MAX; i++)
   printf("\nNota do aluno  %d=%1.1f - Média da turma=%1.2f",
            i,notas[i], media);

  printf("\n\nFim do programa");
  getch();
}
```

7.1.1. Manipulação de vetores

Toda linguagem de programação só manipula tipos básicos

➔ <u>Declaração</u>

```
int votos[500];
```

➔<u>Atribuição</u>

```
votos[2] = 0;
votos[2] += 1;
votos[4] = votos[3]+votos[2];
++votos[2];
```

➔ Inicialização

```
for (i=0; i<500; votos[i++]=0);
```

➔ Leitura

```
for (i=0; i<500; i++)
{  printf("\nDigite o elemento votos[%d]= ",i);
   scanf("%d",&votos [i]);
}
```

➔ Impressão

```
for (i=0; i<500; i++)
   printf("\n votos[%d]=%d  ",i,votos[i]);
```

Exemplo 7.2:

Um armazém trabalha com 100 mercadorias diferentes identificadas de 1 a 100. O dono do armazém anota a quantidade de cada mercadoria vendida durante o mês. Ele tem uma tabela com os preços de venda de cada mercadoria. Escreva um programa para calcular seu faturamento mensal.

```
/*vetor2.c: calcula o faturamento de um armazém */

#include <stdio.h>
#include <conio.h>
#define MAX  100
float preco[MAX], quant[MAX], fatura=0;
int i;

void main(void)
{ clrscr();
  printf("\nCalcula o faturamento de um armazém");

  for (i=0; i<MAX; i++)
  { printf("\nDigite o preço da mercadoria %d:",i);
    scanf("%f",&preco[i]);
    printf("\nDigite a quantidade da mercadoria %d:",i);
    scanf("%f",&quant[i]);
    fatura += (preco[i]*quant[i]);
  }

  printf("\nO faturamento do mes foi R$%5.2f",fatura);

  printf("\nFim do programa");
  getch();
 }
```

Exemplo 7.3:

Dado um vetor de 12 elementos, verificar se existe um elemento igual a *chave*. Se existir, imprimir a posição onde foi encontrado; senão imprimir uma mensagem dizendo que não existe. O vetor e a chave serão lidos.

```
/* vetor3.c: localização de um elemento no vetor */

 #include <stdio.h>
 #include <conio.h>
 #define TAMVET 12
 int vet[TAMVET], i, chave, achou;

 void main(void)
 { printf("\n Localiza um elemento num vetor");
   printf("\n Digite os elementos do vetor: \n");
   for (i=0; i<TAMVET; i++)
    { printf("\nvet[%d]= ",i);
      scanf("%d",&vet[i]);
    }
   printf("\nChave a ser procurada (flag=-1): ");
   scanf("%d",&chave);
   while (chave != -1)
   { achou = 0;  /*indica que não achou ainda */
     i = 0;
     while ( (achou==0) && (i<TAMVET) )
     { if (vet[i] == chave)
       { achou = 1;   /* indica que achou a chave */
         printf("\nA chave %d está na posição %d",chave,i);
       }
       else
         ++i;
     }
     if (achou == 0)
       printf("\nA chave não foi encontrada");
     printf("\nChave a ser procurada (flag=-1): ");
     scanf("%d",&chave);
   }
   printf("\nFim do programa");
   getch();
}
```

Lista de Exercícios Nº 7.1

1) Fazer um programa que leia uma série de 10 dados que representam a altura das pessoas, armazene-os no vetor ALTURA e imprima o vetor, a maior altura e o número de pessoas que possuem esta altura.

2) Fazer um programa que leia um vetor de 20 letras e imprima este vetor em ordem contrária da que foi lida.

3) Fazer um programa que leia um vetor A de 10 elementos reais e construa e imprima um outro vetor B formado da seguinte maneira:
- os elementos das posições pares (2,4,...,10) são os correspondentes de A divididos por 2
- os elementos das posições ímpares (1,3,..9) são os correspondentes de A multiplicados por 3

4) Fazer um programa que leia um vetor A de 20 elementos inteiros e calcule e imprima o valor de S, onde:

$$S = (A[1] - A[20])^2 + (A[2] - A[19])^2 + \dots + (A[10] - A[11])^2$$

5) Modificar o programa que calcula o faturamento do armazém para que o nome das mercadorias sejam também incluídos no programa como dados armazenados num vetor, ou seja, sem serem pedidos para o usuário (vetor2.c).

6) Fazer um programa que leia um vetor com 15 nomes de pessoas. Em seguida leia vários nomes (último nome = ""), um por vez, e pesquise a existência deste nome no vetor e imprima uma mensagem dizendo se o nome foi encontrado ou não. Caso o nome tenha sido encontrado, o algoritmo também deve dizer quantas vezes ele se repetiu.*

7) Fazer um programa que leia vários números inteiros (último número = -1) e imprima o número de vezes que os valores [0..10] foram fornecidos.

7.2. Matrizes Multidimensionais

- Declaração de uma matriz multidimensional:

Sintaxe:

```
tipo nome[tamanho1][tamanho2]...[tamanhon];
```

Exemplos:

```
int id[3][2];      /* Matriz bidimensional de inteiros
                      chamada id com 3 elementos na primeira
                      dimensão e 2 elementos na segunda */

float f[2][3][5];  /*Matriz tridimensional de reais com 2
                     elementos na primeira dimensão, 3
                     na segunda e 5 na terceira */
```

- Inicialização de matrizes multidimensionais:

- Podem ser inicializadas quando da sua declaração.

Exemplos:

```
/* Inicializando todos os elementos */
char id1[3][2] = {{1,2},{3,4},{5,6}};
char id2[3][2] = {1,2,3,4,5,6};
```

- Podem ser inicializadas após serem declaradas.

Exemplo:

```
for(i=0; i<3; i++)
 {   for(j=0; j<2; j++)
     { id[i][j]= 0;
     }
 }
```

Exemplo 7.4:

```
/* Primos.c:     Exibe 4 números primos e calcula os 5 primeiros
                 primos. Mostra o uso de matrizes.         */
#include<stdio.h>
#include<conio.h>

void main(void)
{
     int i,j;
     int num,div;
     int primos1[4]= {7,11,13,17};
     int primos_e_seus_quadrados1[4][2]= {7,49,
                                          11,121,
                                          13,169,
                                          17, 289};
     int primos2[5];
     int primos_e_seus_quadrados2[5][2];

     puts("\n4 numeros primos:");
     for(i=0; i<4; i++)
      printf("Numero primo %d= %d\n",i+1,primos1[i]);

     puts("\n4 numeros primos e seus quadrados:");
     for(i=0;i<4;i++)
    { printf("\nNumero primo %d=%d",
               i+1,primos_e_seus_quadrados1[i][0]);
       printf("\tSeu quadrado= %d\n",
               primos_e_seus_quadrados1[i][1]);
     }

     puts("\nPressione qualquer tecla para continuar...\n");
     getch();
```

```
    puts("\nAgora vamos calcular os numeros primos.");
     primos2[0] = 2;   /* sabemos que 2 e o primeiro primo */
     for(i=0, num = primos2[i]+1; i<5; num++)
    {      div=primos2[i];
           for(j=i;div>=2;div=primos2[--j])
         {  if(!(num%div))
                 break;
              else
              if (div==2)
              {  primos2[++i] = num;
                       break;
                 }
           }
     }

     puts("\nPrimeiros 5 numeros primos:");
     for(i=0;i<5;i++)
           printf("O numero primo %d= %d\n",i+1,primos2[i]);

     puts("\nAgora vamos calcular o quadrado dos numeros primos.");
     for(i=0; i<5; i++)
    {      num = 1;
           for (j=0; j<2; j++)
         {
                 num *= primos2[i];
                 primos_e_seus_quadrados2[i][j] = num;
           }
     }

     puts("\nPrimeiros 5 numeros primos e seus quadrados:");
     for(i=0;i<5;i++)
    {printf("O numero primo %d= %d",
             i+1,primos_e_seus_quadrados2[i][0]);
      printf("\tSeu quadrado=%d\n",
             primos_e_seus_quadrados2[i][1]);
     }

    printf("\nFim do programa");
    getch();
  }
```

7.2.1. Número de elementos e de dimensões das matrizes

As matrizes podem ter n dimensões, mas usualmente utilizam-se matrizes de 2 e 3 dimensões. O número de elementos é o resultante da multiplicação da quantidade de elementos de cada dimensão.

```
int mat1[3][3];
```

mat1 ➔ 2 dimensões   3 * 3 = 9 elementos

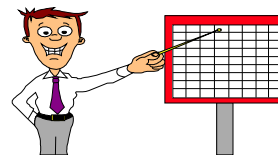
```
float mat2 [3][3][2];
```

mat2 ➔ 3 dimensões   3 * 3 * 2 = 18 elementos

➔ Um exemplo com 3 dimensões poderia ser:

<u>Nº ALUNOS</u> x <u>Nº DISCIPLINAS</u> x <u>Nº PROVAS POR DISCIPLINA</u>

O aluno 2 obteve, na disciplina nº 2, na prova nº 1, nota 8.5

7.2.2. Manipulação de matrizes

➔ Declaração

```
int  mat[4][5];
```

➔ Atribuição

```
mat [0][0] = 0;
mat [2][3] += 2;
mat[2][4]=mat [2][4] + 2;
--mat[0][0];
```

➔ Inicialização

```
for (i=0; i<4; i++)
   for (j=0; j<5; j++)
      mat[i][j] = 0;
```

➔ Leitura { por linhas}

```
for (i=0; i<4; i++)
   for (j=0; j<5; j++)
   {   printf("\n mat[%d][%d]= ",i,j);
       scanf("%d",&mat[i][j]);
    }
```

➔ Impressão { por linhas}

```
for (i=0; i<4; i++)
{   for (j=0; j<5; j++)
        printf ("%d    ",mat[i][j]);
    printf("\n");
}
```

Exemplo 7.5:

Dada a matriz abaixo, faça um programa que a imprima por linhas e por colunas.

	0	1	2	3
0	M	E	S	A
1	A	M	O	R
2	R	A	L	O

```
/* mat1.c: ilustra impressao de matriz por linhas e
           por colunas */

#include <stdio.h>
#include <conio.h>

char mat[3][4]={‘M’,’E’,’S’,’A’,
                ‘A’,’M’,’O’,’R’,
                ‘R’,’A’,’L’,’O’};
int i,j;
void main(void)
{ clrscr();

  printf(“\nMatriz por linhas\n”);
  for (i=0; i<3; i++)
  { for (j=0; j<4; j++)
      printf(“%c   ”,mat[i][j]);
    printf(“\n”);
  }

  printf(“\nMatriz por colunas\n”);
  for (j=0; j<4; j++)
  { for (i=0; i<3; i++)
      printf(“%c   ”,mat[i][j]);
    printf(“\n”);
  }

  printf(“\n Fim do programa”);
  getch();
}
```

Exemplo 7.6:

Dada uma matriz 4x5, faça um programa para somar os elementos de cada linha, gerando o vetor somalinha. Em seguida somar os elementos do vetor na variável **total**.

```
 /* mat2.c: soma as linhas e todos os elementos de
            uma matriz 4x5 */
#include <stdio.h>
#include <conio.h>
int mat[4][5]={7,3,4,5,6,
               1,0,4,9,-1,
               2,3,-2,4,6,
               1,5,-4,0,8};
int somalinha[4];
int i,j,total=0;
void main(void)
{ clrscr();
  for (i=0; i<4; i++)
  { somalinha[i] = 0;
    for (j=0; j<5; j++)
      somalinha[i] += mat[i][j];
    total += somalinha[i];
  }
  printf("\nSoma das linhas da matriz\n");
  for (i=0; i<4; i++)
  { for (j=0; j<5; j++)
      printf("%d   ",mat[i][j]);
    printf("\tTotal da linha %d=%d\n",i,somalinha[i]);
  }
  printf("\nTotal dos elementos da matriz=%d ",total);
  printf("\n\nFim do programa");
  getch();
}
```

Exemplo 7.7:

Dado um tabuleiro de xadrez onde, para facilitar a identificação das peças, estão codificados:

1- peões   2- cavalos   3- torres
4- bispos   5- reis   6- rainhas   0- vazio

faça um programa para:

1) <u>contar a quantidade de cada tipo de peça no tabuleiro</u>

```
/* mat3.c: conta a quantidade de pecas num tabuleiro de xadrez*/
#include <stdio.h>
#include <conio.h>
int xadrez[8][8]={1,3,4,5,6,0,2,2,
                  2,1,0,0,1,3,0,0,
                  2,3,2,4,6,0,0,3,
                  1,5,4,0,3,1,1,1,
                  0,0,3,3,4,1,2,2,
                  3,0,0,0,2,1,1,6,
                  4,0,0,0,1,2,1,5,
                  3,6,5,1,4,4,5,2};
char *pecas[7]={"vazio",
                "peões",
                "cavalos",
                "torres",
                "bispos",
                "reis",
                "rainhas"};
int conta[7];
int i,j;
void main(void)
{ clrscr();
  for (i=0; i<7; conta[i++]=0);
  printf("\nTabuleiro de xadrez:\n");
  for (i=0; i<8; i++)
  { for (j=0; j<8; j++)
    { if (xadrez[i][j]>=0 && xadrez[i][j]<=6)
        ++conta[xadrez[i][j]];
        printf("%d   ",xadrez[i][j]);
    }
    printf("\n");
  }
  printf("\nTotal de peças no tabuleiro de xadrez:");
  for (i=1; i<7; i++)
    printf("\nTotal de %d-%s= %d ",i,pecas[i],conta[i]);

  printf("\n\nFim do programa");
  getch();
}
```

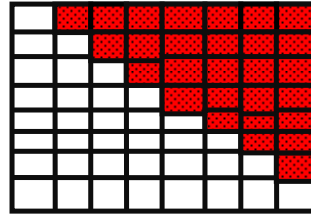
Exemplo 7.8:

2) Contar o número de cada tipo de peça na região do tabuleiro abaixo da diagonal principal

```
printf("\nTabuleiro de xadrez abaixo da
          diagonal principal:");
for (i=1; i<8; i++)
  for (j=0; j<=i-1; j++)
     if (xadrez[i][j] >= 0 && xadrez[i][j] <= 6)
       ++conta[xadrez[i][j]];
```

```
/* mat4.c: conta a quantidade de pecas num tabuleiro de xadrez
     abaixo da diagonal principal*/
#include <stdio.h>
#include <conio.h>

int xadrez[8][8]={1,3,4,5,6,0,2,2,
                  2,1,0,0,1,3,0,0,
                  2,3,2,4,6,0,0,3,
                  1,5,4,0,3,1,1,1,
                  0,0,3,3,4,1,2,2,
                  3,0,0,0,2,1,1,6,
                  4,0,0,0,1,2,1,5,
                  3,6,5,1,4,4,5,2};

char *pecas[7]={"vazio", "peoes", "cavalos", "torres", "bispos",
        "reis", "rainhas"};
int conta[7];
int i,j;

void main(void)
{ clrscr();
  for (i=0; i<7; conta[i++]=0);

  printf("\nTabuleiro de xadrez abaixo da diagonal principal:\n");
  for (i=1; i<8; i++)
  { for (j=0; j<=i-1; j++)
    { if (xadrez[i][j]>=0 && xadrez[i][j]<=6)
        ++conta[xadrez[i][j]];
        printf("%d   ",xadrez[i][j]);
    }
    printf("\n");
  }

  printf("\nTotal de pecas no tabuleiro de xadrez:");
  for (i=1; i<7; i++)
    printf("\nTotal de %d-%s= %d ",i, pecas[i],conta[i]);

  printf("\n\nFim do programa");
  getch();
}
```

3) Idem 2, mas para os elementos acima da diagonal principal

```
for (i=0; i<=6; i++)
 for (j=i+1; j<8; j++)
    if (xadrez[i][j] >= 0 &&
        xadrez[i][j] <= 6)
      ++conta[xadrez[i][j]];
```

Lista de Exercícios Nº 7.2

1) Fazer um programa que gere as três matrizes seguintes.

mat1

	X	X	X	X	X	X	X
X		X	X	X	X	X	X
X	X		X	X	X	X	X
X	X	X		X	X	X	X
X	X	X	X		X	X	X
X	X	X	X	X		X	X
X	X	X	X	X	X		X
X	X	X	X	X	X	X	

mat2

X	X	X	X	X	X	X	X
X	X	X			X	X	X
X	X	X			X	X	X
X							X
X							X
X	X	X			X	X	X
X	X	X			X	X	X
X	X	X	X	X	X	X	X

mat3

			X	X			
		X	X	X	X		
	X	X	X	X	X	X	
X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X

Sugestões:

1) ***Inicialize as matrizes com ´ ´ ou com ´X´, de acordo com o que achar melhor. Exemplo:***

```
char mat1[8][8], mat2[8][8], mat3[8][8];
int i, j;

for (i=0; i<8; i++)
  for (j=0; j<8; j++)
     mat1[i][j]='X';
```

2) ***Imprima as matrizes desta forma:***

```
printf("\nMATRIZ 1\n");
for (i=0; i<8; i++)
{ for (j=0; j<8; j++)
    printf("%c  ",mat1[i][j]);
  printf("\n");
}
getch();
```

2) *Fazer um programa que gere a seguinte matriz:*

1	1	1	1	1	1
1	2	2	2	2	1
1	2	3	3	2	1
1	2	3	3	2	1
1	2	2	2	2	1
1	1	1	1	1	1

3) A matriz a seguir contém em cada linha as 5 notas de provas obtidas pelos N alunos de uma classe. O índice da linha corresponde ao número do aluno e o índice da coluna ao número da prova.

Faça um programa que leia o número de alunos N, leia as 5 notas de cada um dos N alunos, armazene-as numa matriz e calcule e imprima, para cada aluno, a sua média e a média geral da turma . Sugestão: utilize um vetor para calcular as médias.

4) A distância entre várias cidades é dada pela tabela abaixo (em Km):

-	90	45	120	200
90	-	450	500	95
45	450	-	600	240
120	500	600	-	950
200	95	240	950	-

Sabendo que as cidades são identificadas por 1, 2, 3, 4 e 5, faça um programa que:

a) crie a tabela em memória utilizando uma matriz

b) leia vários pares de cidades (1 a 5) e imprima a distância entre elas (flag = 0)

5) A matriz seguinte representa o pátio de um depósito de uma empresa de construções, que utiliza as seguintes convenções:

201	000	405	234
599	777	345	566
424	189	199	124
675	309	000	736

1- cimento 2-areia
3- madeiras 4-tubos
5- blocos de concreto
6- cal 7- saibro

Em cada parte do depósito estão representados o código do material e a quantidade armazenada do material. Por exemplo, (199) = (1) cimento + (99) 99 sacos. Escrever um programa capaz de contar e imprimir quantos elementos de cada material existe no depósito.

8. Strings

- Strings nada mais são que vetores de caracteres, cujo último elemento é igual a um nulo ('\0')

Exemplos:

```
char str[9]= "Bom dia!";
char str[]= "Bom dia!";
char str[]= {'B','o','m',' ','d','i','a','!','\0'};
```

Exemplo 8.1:

```
/* string1.c: Mostra o uso de strings    */
#include<stdio.h>
#include<conio.h>

void main(void)
{ char str1[15] = "Primeiro nome:";
  char str2[]   = "Sobrenome:";
  char str3[]   = {'M','u','i','t','o',
                   ' ','O','b','r','i','g','a','d','a','!','\0'};
  char prim_nome[50];
  char sobrenome[50];
  int cont;
  char ch;

  puts("Complete as informacoes:\n");
  puts(str1);
  gets(prim_nome);
  puts(str2);
  for(cont=0;(ch=getche())!='\r';cont++)
     sobrenome[cont] = ch;

  sobrenome[cont] = '\0';

  for(cont=0; prim_nome[cont]!='\0'; cont++);

  printf("\nSeu primeiro nome tem %d letras\n",cont);

  for(cont=0; sobrenome[cont]!='\0'; cont++);

  printf("\nSeu sobrenome tem %d letras\n",cont);
  printf(str3);

  printf("\nFim do programa");
  getch();
 }
```

8.1. Funções especiais para tratamento de strings: biblioteca <string.h>

- Toda operação que envolve strings deve ser feita através de funções especiais que se encontram na biblioteca <string.h>. As principais funções são:

```
strcat (destino, fonte);
      anexa o string <fonte> ao string <destino>

strcpy (destino, fonte);
     copia o string <fonte> para o string <destino>

strcmp (string1,string2);
      compara o <string1> com o <string 2> pela ordem
      alfabética (conforme tabela ASCii), resultando em:
         menor que 0 → <string1>  menor que <string2>
         igual a 0 → <string1>  igual a <string2>
         maior que 0 → <string1>  maior que <string2>

strchr(string, caracter);
      verifica se o <caracter> se encontra na <string> e retorna a
      posição da primeira ocorrência do <caracter> no <string>; se o
      <caracter> não for encontrado, retorna NULL

tamanho = strlen(string);
         retorna o <tamanho> de uma <string>  em número de
         caracteres.

destino= strrev(string);
        retorna para <destino> a  <string> invertida.

destino = strupr(string);
       retorna para <destino> a <string> convertida para maiúscula.

destino = strlwr(string);
       retorna para <destino> a <string> convertida para minúscula.
```

Exemplo 8.2:

```
/*strcat.c: ilustra o uso da funcao strcat */
#include <string.h>
#include <stdio.h>
#include <conio.h>

void main(void)
{  char destino[25];
   char *espaco = " ",
    *segundo = "Segundo string",
    *primeiro = "Primeiro string";

   clrscr();
   strcpy(destino, primeiro);
   strcat(destino, espaco);
   strcat(destino, segundo);
   printf("%s\n", destino);
   getch();
}
```

Exemplo 8.3:

```
/*strcmp.c: ilustra o uso da funcao strcmp */

#include <conio.h>
#include <string.h>
#include <stdio.h>

int main(void)
{
    char *str1 = "aaa", *str2 = "bbb", *str3 = "ccc", *str4="aaa";
    int result;

    clrscr();
    printf("String1=%s String2=%s String3=%s String4=%s\n\n",
            str1,str2,str3,str4);
    printf("\nComparando String1 e String2");
    result = strcmp(str2, str1);
    if (result > 0)
       printf("\nString 2 e maior que string 1 \n");
    else  if (result < 0)
             printf("\nString 2 e menor que string 1 \n");
          else
             printf("\nString 2 e string 1 sao iguais\n");

    printf("\nComparando String1 e String3");
    result = strcmp(str3, str1);
    if (result > 0)
       printf("\nString 3 e maior que string 1 \n");
    else  if (result < 0)
             printf("\nString 3 e menor que string 1 \n");
          else
             printf("\nString 3 e string 1 sao iguais\n");

    printf("\nComparando String2 e String3");
    result = strcmp(str2, str3);
    if (result > 0)
       printf("\nString 2 e maior que string 3 \n");
    else  if (result < 0)
            printf("\nString 2 e menor que string 3 \n");
          else
            printf("\nString 2 e string 3 sao iguais\n");

    printf("\nComparando String1 e String4");
    result = strcmp(str1, str4);
    if (result > 0)
       printf("\nString 1 e maior que string 4 \n");
    else  if (result < 0)
             printf("\nString 1 e menor que string 4 \n");
          else
             printf("\nString 1 e string 4 sao iguais\n");
    printf("\nFim do programa");
    getch();
 }
```

Exemplo 8.4:

```
/* strchr.c: ilustra a utilizacao da funcao strchr */

#include <string.h>
#include <stdio.h>
#include <conio.h>

void main(void)
 {
    char string[15];
    char *result, caracter;

    clrscr();


    puts("Digite uma frase: \n");
    gets(string);

    for ( ; ; )
    { printf("\nDigite um caracter (. para terminar): ");
       caracter = getche();

       if (caracter == '.')
          break;

       result = strchr(string, caracter);
       if (result != NULL)
          printf("\nO caracter %c se encontra na posicao: %d\n",
             caracter, result-string);
       else
          printf("\nO caracter %c nao se encontra na frase",
               caracter);
    }

    printf("\n\n Fim do programa");
    getch();
 }
```

Exemplo 8.5:

```
/*strcpy.c: ilustra o uso da funcao strcpy */

#include <stdio.h>
#include <conio.h>
#include <string.h>

void main(void)
 {
    char destino[15]="Frase inicial";
    char fonte[15] = "QUALQUER FRASE";
    clrscr();

    printf("ANTES DA COPIA: ");
    printf("\nString fonte: %s string destino: %s ",   fonte, destino);
    strcpy(destino,fonte);
    printf("\n\nDEPOIS DA COPIA");
    printf("\nString fonte: %s string destino: %s ",   fonte,destino);

    printf("\n\nFim do programa");
    getch();
 }
```

Lista de Exercícios nº 8

1. *Escreva um programa que, fornecidas duas strings pelo usuário, as concatena em uma terceira string, exibindo-a na tela.*

2. *Faça um programa que leia várias frases e conte e imprima quantas palavras há em cada frase. Flag= frase em branco.*

3. *Faça um programa que leia e organize uma lista de nomes e telefones de várias pessoas. Depois leia o nome das pessoas e forneça o número de seu telefone. Utilize os flags que achar mais adequados.*

9. Estruturas (ou Registros)

- São agrupamentos de variáveis que podem ser de tipos diferentes, mas são referenciadas por um nome comum.

9.1. Definição de uma estrutura

Sintaxe:

```
struct [nome]  { tipo nome_variável1;
                 tipo nome_variável2;
                      . . .

                 tipo nome_variáveln;
               } [lista_de_variáveis];

typedef struct { tipo nome_variável1;
                              tipo nome_variável2;
                                      . . .

                              tipo nome_variáveln;
                } [nome_tipo];
```

Exemplos:

```
struct informacoes
      { char nome[30];
        int dia_nascimento;
        int mes_nascimento;
        int ano_nascimento;
      };

struct informacoes info1, info2;

struct notas
      { char nome_aluno[30];
        float nota_prova1;
        float nota_prova2;
        float media;
      } nota1, nota2;

typedef struct { char info[20];
                 struct no *esq, *dir;
               } no;
no no1, no2, *apno1, *apno2;
```

9.2. Declaração de variáveis do tipo estrutura

- Da mesma forma como se declara uma variável como sendo de qualquer um dos tipos básicos, pode-se declarar uma variável como sendo do tipo estrutura. E pode-se declarar uma estrutura como sendo um novo tipo.

- Definição da estrutura e declaração de variável

Exemplo:
```
struct informacoes
       { char nome[30];
         int dia_nascimento;
         int mes_nascimento;
         int ano_nascimento;
       } inf_sobre_mim;
```

- Definição de um tipo estrutura

Exemplo:
```
typedef struct {  char nome[30];
                  char cpf[12];
                  char rg[12];
                  int idade;
                } ficha;
```

- Declaração de variáveis estrutura após sua definição

Exemplo:
```
struct informacoes inf_sobre_voce;
ficha ficha1, ficha2;
```

9.3. Referência aos campos de uma estrutura

- Campos de uma estrutura são referenciados utilizando-se o chamado "operador ponto".

Sintaxe:
```
nome_da_estrutura.nome_do_campo
```

Exemplo:
```
    gets(inf_sobre_mim.nome);
    scanf("%d",&inf_sobre_voce.dia_nascimento);
    ficha1.idade = 18;
    strcpy(ficha2.cpf, "123456789-10");
```

9.4. Inicialização de estruturas

- Estruturas podem ser inicializadas dispondo-se os valores a serem atribuídos a cada campo entre chaves, na ordem de declaração dos campos dentro da estrutura.

Exemplo:

```
struct informacoes inf_sobre_mim = {"Elisa",
                                     25,
                                     12,
                                     1970
                                    };
```

- Podem ser inicializadas campo a campo.

Exemplo:

```
strcpy(inf_sobre_voce.nome, "Diego");
inf_sobre_voce.dia_nascimento = 25;
inf_sobre_voce.mes_nascimento = 1;
inf_sobre_voce.ano_nascimento = 1990;
```

9.5. Atribuição entre estruturas

- Tendo-se duas estruturas de um mesmo tipo, uma pode ser atribuída a outra, sem que seja necessário fazer a atribuição campo a campo.

Exemplo:

```
inf_sobre_voce = inf_sobre_mim;
```

equivale a:

```
strcpy(inf_sobre_voce.nome,inf_sobre_mim.nome);
inf_sobre_voce.dia_nascimento=inf_sobre_mim.dia_nascimento;
inf_sobre_voce.mes_nascimento=inf_sobre_mim.mes_nascimento;
inf_sobre_voce.ano_nascimento=inf_sobre_mim.ano_nascimento;
```

9.6. Estruturas Aninhadas

- Um campo de uma estrutura pode ser ele próprio uma estrutura.

Exemplos:

```
struct reta
        { struct ponto pt1;
          struct ponto pt2;
        }; /*Uma reta é definida por dois pontos*/
struct ponto
        {int x;
         int y;
        };
```

```
struct informacoes
      { char nome[80];
        struct { int dia;
                 int mes;
                 int ano;
               } data_nascimento;
      };
```

9.6.1. Inicialização de estruturas aninhadas

Exemplos:

```
struct reta reta1, reta2;

reta1.pt1.x= reta1.pt2.y=10;
reta1.pt1.y= reta1.pt2.x=0;
reta2=reta1;
```

9.7. Vetores de Estruturas

- Declaração de vetores cujos elementos são do tipo estrutura:

- Da mesma forma como se declara um vetor de qualquer um dos tipos básicos, pode-se declarar um vetor de estrutura.

Exemplo:

```
struct informacoes inf_sobre_todos[10];
```

- Inicialização de vetores de estruturas:

- Podem ser inicializados dispondo-se os valores a serem atribuídos a cada campo da estrutura de cada posição do vetor entre chaves.

Exemplo:

```
struct informacoes
         { char nome[80];
             struct  {    int dia;
                          int mes;
                          int ano;
                    } data_nascimento;
          }inf_sobre_todos[3]= { {"Jose", 25, 1, 1970},
                                 {"Maria",16, 7, 1990},
                                 {"João", 11, 9, 2001},
                             };
```

- Podem ser inicializadas campo a campo

Exemplos:

```
strcpy(inf_sobre_todos[0].nome,"Jose");
inf_sobre_todos[0].data_nascimento.dia = 25;
inf_sobre_todos[0].data_nascimento.mes = 1;
inf_sobre_todos[0].data_nascimento.ano = 1970;

strcpy(inf_sobre_todos[1].nome, "Maria");
inf_sobre_todos[1].data_nascimento.dia = 16;
inf_sobre_todos[1].data_nascimento.mês = 7;
inf_sobre_todos[1].data_nascimento.ano = 1990;
```

Exemplo 9.1:

```
/* struct.h: arquivo cabecalho de struct.c */
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

#define SAIR           1
#define CORRIGIR       2
#define MONTALISTA     3
#define VERINFORM      4

void corrigeInformacoes(void);
int montaLista(void);
void verInform(int);

struct informacoes
       {   char nome[50];
           char fone[20];
           struct { int dia;
                    int mes;
                    int ano;
                  } data_nascimento;
       }inf_sobre_voce={"Clotilde","3855-6535",11,2,2000};
struct informacoes agenda[20];
```

```
/* Struct.c: Ilustra o uso de estruturas e vetores de de estruturas */
#include "struct.h"
void main(void)
{ int opcao;
  int numero_de_amigos;

  clrscr();
  puts("***********************************************");
  printf("Seu nome e %s.\n",inf_sobre_voce.nome);
  printf("Seu telefone e %s,\n", inf_sobre_voce.fone);
  printf("A data em que nasceu foi %d/%d/%d.\n",
         inf_sobre_voce.data_nascimento.dia, inf_sobre_voce.data_nascimento.mes,
         inf_sobre_voce.data_nascimento.ano);
  puts("***********************************************");
```

```
for(;;)
  { do
      {puts("\nEscolha uma das opcoes:\n");
       puts("1) Sair do programa.");
       puts("2) Corrigir informacoes sobre voce.");
       puts("3) Montar uma pequena lista com informacoes sobre seus amigos.");
       puts("4) Ver as informacoes contidas na sua lista.");
       scanf("%d",&opcao);
       fflush(stdin);
     } while(opcao<1 || opcao>4);


         switch(opcao)
         {  case SAIR:
                  exit(0);
                  break;
            case CORRIGIR:
                  corrigeInformacoes();
                  break;
            case MONTALISTA:
                  numero_de_amigos = montaLista();
                  break;
            case VERINFORM:
                  verInform(numero_de_amigos);
                  break;
            default:
                  puts("Voce nao escolheu uma opcao valida!");
          }
      }
   }

  void corrigeInformacoes(void)
  {
  puts("Forneca o seu nome:");
  gets(inf_sobre_voce.nome);
  puts("Forneca seu telefone:");
  gets(inf_sobre_voce.fone);
  puts("Forneca o dia do seu nascimento:");
  scanf("%d",&inf_sobre_voce.data_nascimento.dia);
  puts("Forneca o mes do seu nascimento:");
  scanf("%d",&inf_sobre_voce.data_nascimento.mes);
  puts("Forneca o ano do seu nascimento:");
  scanf("%d",&inf_sobre_voce.data_nascimento.ano);
  fflush(stdin);
  puts("***********************************************");
  printf("Seu nome e %s.\n",inf_sobre_voce.nome);
  printf("Seu telefone e %s,\n",inf_sobre_voce.fone);
  printf("A data em que nasceu foi %d/%d/%d.\n",
          inf_sobre_voce.data_nascimento.dia,
          inf_sobre_voce.data_nascimento.mes,inf_sobre_voce.data_nascimento.ano);
  puts("***********************************************");
  }
```

```
int montaLista(void)
  {   static int conta_amigo = 0;
      char resposta;

      do{   puts("Deseja entrar com informacoes sobre algum amigo?(s/n)");
            resposta = getch();
            if(resposta=='s' || resposta=='S')
            {puts("Forneca o nome do seu amigo:");
             gets(agenda[conta_amigo].nome);
             puts("Forneca o telefone do seu amigo:");
             gets(agenda[conta_amigo].fone);
             puts("Forneca o dia do nascimento do seu amigo:");
             scanf("%d",&agenda[conta_amigo].data_nascimento.dia);
             puts("Forneca o mes de nascimento do seu amigo:");
                   scanf("%d",&agenda[conta_amigo].data_nascimento.mes);
             puts("Forneca o ano do nascimento do seu amigo:");
             scanf("%d",&agenda[conta_amigo].data_nascimento.ano);
             conta_amigo++;
       }
      fflush(stdin);
      }while(resposta=='s' || resposta=='S');
      return(conta_amigo);
  }


  void verInform(int numero_de_amigos)
  {
      int i;
      puts("Informacoes sobre seus amigos.\n");
      for(i=0; i<numero_de_amigos; i++)
      {
            printf("\nNome: %s\n", agenda[i].nome);
            printf("Telefone: %s\n", agenda[i].fone);
            printf("Data de nascimento: %d/%d/%d\n",
                    agenda[i].data_nascimento.dia,agenda[i].data_nascimento.mes,
                    agenda[i].data_nascimento.ano);
      }
  }
```

9.8. Exercício Prático 1

Defina as estruturas mostradas nas figuras e inicialize-as:

ficha

nome	
salario	
idade	sexo

```
struct f {  char nome[30];
            float salario;
            int idade;
            char sexo;
          } ficha = {"Luiza",3495.00,25,'f'};
```

func

<table>
<tr><td colspan="3">nome</td></tr>
<tr><td colspan="3">endereco</td></tr>
<tr><td>rua</td><td>numero</td><td>cep</td></tr>
<tr><td>cidade</td><td colspan="2">estado</td></tr>
<tr><td colspan="3">salario</td></tr>
</table>

```
struct dados { char nome[40];
               struct {char rua[30];
                       int numero;
                       char cep[9];
                      } endereco;
               char cidade[30], estado[20];
               float salario;
             } func;
```

A atribuição de valores à estrutura func poderia ser assim:

```
strcpy(func.nome,"José João da Silva");
strcpy(func.endereco.rua,"Rua Tremembé");
strcpy(func.cidade,"Itamarandiba");
strcpy(func.estado,"Minas Gerais");
func.endereco.numero = 12;
func.salario = 1005.00;
strcpy(func.endereco.cep,"30045-000");
```

```
struct reg {char *nome;
            long cpf, rg;
            float horastrabalhadas[7];
            float salario;
            float fgts[2][2];
            } regpag;
```

```
regpag.cpf = 1234567;
regpag.salario = 7500.00;
strcpy(regpag.nome,"Joaquim Manoel de Mendonça");
regpag.rg = 12377781;

for (i=1; i<=6; regpag.horastrabalhadas[i++]=0);

for (i=0; i<2; i++)
  for (j=0; j<2; j++)
    regpag.fgts[i][j]=0;
```

9.9. Exercício Prático 2

O que representa a definição a seguir?

```
struct reg  { char nome [30];
              long cpf;
              float salario;
              char cargo[30];
            } vetreg[1000];
```

Exemplo 9.2

Um pequeno município deseja cadastrar seus contribuintes proprietários de imóveis, verificar se seu IPTU está em dia e calcular a multa daqueles que estão em débito. Fazer um programa para resolver o problema da prefeitura, considerando que:

- serão fornecidos o nome do proprietário, o número do imóvel, o valor do IPTU e os meses em atraso (último nome = "fim")
- a multa deve ser calculada de acordo com a tabela abaixo, que deve estar armazenada em memória:

valor do iptu	% por mês em atraso
até 200,00	1%
de 200,01 a 400,00	3%
de 400,01 a 700,00	5%
de 700,01 a 1000,00	8%
acima de 1000,00	10%

```
 /*Struct1.c: calcula a multa sobre o  IPTU de contribuintes -
   ilustra o uso de vetores de estruturas */
#include <stdio.h>
#include <conio.h>

struct tab {float de, ate, porc;
           } tabela[5] = {0.0,     200.00,     0.01,
                         200.01,  400.00,     0.03,
                         400.01,  700.00,     0.05,
                         700.01,  1000.00,    0.08,
                         1000.01, 1000000.00, 0.1,
                        };

struct d { char nome[30];
           int imovel;
           float iptu;
           int atraso;
         } dados;

float  multa;
int i, achou;

void main(void)
{    clrscr();
     printf("\nDados do contribuinte");
     printf("\n\nNome: ");
     scanf("%s",dados.nome);
     while (strcmp(dados.nome,"fim") != 0)
     {    printf("\nNumero do imovel: ");
          scanf("%d",&dados.imovel);
          printf("\nValor do IPTU: ");
          scanf("%f",&dados.iptu);
          printf("\nMeses atrasados: ");
          scanf("%d",&dados.atraso);

          achou = 0;
          if  (dados.atraso == 0)
          {  multa = 0;
             achou = 1;
             printf("\n O contribuinte: %s nao esta em atraso e nao tem multa a
                      pagar.", dados.nome);
          }

          /* localiza a faixa de iptu do contribuinte */
          i = 0;
          while (achou == 0)
          {   if (dados.iptu >= tabela[i].de && dados.iptu <= tabela[i].ate)
              {   multa = dados.iptu*dados.atraso*tabela[i].porc;
                  achou = 1;
                  printf("\n O contribuinte: %s tem %d meses em atraso ",
                         dados.nome,dados.atraso);
                  printf("\n e pagara uma multa de R$%.2f calculada com %.0f %%
                           de juros/mes ", multa,tabela[i].porc*100);
              }
              else  ++i;
          }
          printf("\n");
          getch();
          clrscr();
          printf("\nDados do contribuinte");
          printf("\n\nNome (fim para terminar): ");
          scanf("%s",dados.nome);
     }
     printf("\nFim do programa... pressione <enter>");
     getch();
 }
```

Lista de Exercícios nº 9

***1.** Escreva um programa em que é definido um vetor de registros com 5 campos. Cada posição do vetor deve consistir em uma estrutura com informações sobre alunos: o nome, a idade, as notas de duas provas e a média destas provas. Forneça ao usuário as seguintes opções:*

- *Entrar com informações sobre os alunos: nome, idade e as notas das duas provas;*
- *Ver os valores que foram fornecidos;*
- *Calcular e ver a média de cada aluno;*
- *Calcular e ver a média da classe.*

2) Fazer um programa que leia vários nomes e notas de alunos (flag= nome= ´ULTIMO´), utilizando uma estrutura para armazenar os dados lidos, e calcule e imprima o conceito obtido pelos alunos, de acordo com o seguinte critério:

Notas de 0 a 3 - conceito I (Insuficiente)
de 3.1 a 6 - conceito R (Reprovado)
de 6.1 a 7.5 - conceito C
de 7.6 a 8.5 - conceito B
de 8.6 a 10 - conceito A

3) Uma agência de matrimônios cadastra seus clientes e seus dados pessoais da seguinte maneira:

código	nome	respostas sobre hábitos e gostos pessoais
0 = moça 1 a 9 = rapaz	até 40 caracteres	1 = sim 2 = indiferente 3 = não

Estes dados, frutos de respostas de clientes a um questionário, são armazenados num arquivo organizado de forma que um registro com o campo código=0 é seguido por 9 registros com campo código=1, 2, ... até 9, e assim sucessivamente, ou seja, uma moça seguida de 9 rapazes pretendentes.

Faça um programa que automatize o processo de seleção do par ideal da agência. Para isto, o programa deve ler o arquivo de dados e, para cada um dos 9 rapazes de uma seqüência, deve calcular os índices de afinidade e de incompatibilidade com a moça que o antecede. Estes índices são encontrados comparando-se as 10 respostas do rapaz com as respectivas respostas da moça. Se as respostas coincidem, soma-se 1 à afinidade, caso contrário, soma-se 1 à incompatibilidade. Além disto, deve ser calculada a diferença líquida entre os 2 índices (afinidade - incompatibilidade).

Os resultados devem ser impressos, indicando o nome da moça e dos 9 rapazes pretendentes (junto com o valor de seus 2 índices). Ao final de cada seqüência deve ser indicado o nome do rapaz que mais se afina com a moça.

10. Arquivos

O conceito de *Stream*:

- Para fornecer ao programador uma interface independente de dispositivo, o sistema de E/S do C trabalha com um conceito abstrato denominado ***stream***, que consiste em um dispositivo lógico, o qual é associado ao dispositivo físico, denominado **"arquivo".** Um **"arquivo"**, por sua vez, pode ser desde um arquivo propriamente dito, até um terminal ou uma impressora.

10.1. Declaração de Arquivos

O ponteiro de arquivo:

- A maioria das funções tem como argumento um ponteiro de arquivo, que permite o acesso a várias informações sobre o arquivo: seu nome, status, posição atual, etc, identificando um arquivo específico em disco.

- É usado pela ***stream*** associada para direcionar as operações das funções de E/S.

- É uma variável ponteiro do tipo **FILE** (definido em stdio.h).

Exemplo:

```
FILE *fp;          /* Declara uma variável
                      ponteiro de arquivo */
```

10.2. A função `fopen( )`

- Arquivo de cabeçalho a ser incluído: **stdio.h**

- Abre uma ***stream*** e associa um arquivo a ela.

- Recebe como argumentos o nome do arquivo e o modo em que ele deve ser aberto.

- Retorna um ponteiro para arquivo ou, se ocorrer alguma falha, retorna um **NULL** (definido em **stdio.h** como '\0')

Exemplo:

```
FILE *fp;
if ((fp = fopen( "teste.txt", "r") ) == NULL)
{  printf("Arquivo não pode ser aberto!\n");
   exit(1);
}
/* Abre um arquivo texto no modo de leitura, mas se ocorrer algum
   erro na abertura do arquivo, aborta */
```

Tabela 10.1. Valores para o argumento modo de `fopen()`

Modo	*Significado*
`r`	Abre um arquivo texto para leitura.
`w`	Cria um arquivo texto para escrita.
`a`	Anexa a um arquivo texto.
`rb`	Abre um arquivo binário para leitura.
`wb`	Cria um arquivo binário para escrita.
`ab`	Anexa a um arquivo binário.
`r+`	Abre um arquivo texto para leitura/escrita.
`w+`	Cria um arquivo texto para leitura/escrita.
`a+`	Anexa ou cria um arquivo texto para leitura/escrita.
`r+b`	Abre um arquivo binário para leitura/escrita.
`w+b`	Cria um arquivo binário para leitura/escrita.
`a+b`	Anexa a um arquivo binário para leitura/escrita

10.3. A função `fclose( )`

- Arquivo de cabeçalho a ser incluído: **stdio.h**
- Fecha uma ***stream*** que foi aberta através de uma chamada a **fopen()**.
- Se bem sucedida retorna o valor zero.
- Um valor de retorno diferente de zero indica ocorrência de erro, podendo ser utilizada a função **ferror()** para se determinar o erro ocorrido.

Exemplo:

```
FILE *fp;
if ( (fp = fopen( "teste.txt", "w") ) == NULL)
{
     printf("Arquivo não pode ser aberto!\n");
     exit(1);
}
. . .
fclose(fp);
```

Obs: Em geral, o sistema operacional impõe um limite ao número de arquivos que podem ser simultaneamente abertos, por isso deve-se fechar um arquivo ao se encerrar o seu uso.

Exemplo 10.1

```
/* FILE1.c - Ilustra a criação de arquivos do tipo texto */

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

void main()
{
      FILE *arq;
     char nome[30], nomearq[30];
     int idade, idadearq;
     float altura, alturaarq;

      arq = fopen(“teste.txt”, "w");
     if (arq == NULL)
     { printf("\nErro na criação do arquivo teste.txt");
       getch();
        exit(1);
      }

     puts("Qual é o seu nome?  ");
     gets(nome);
     puts("Qual é o sua idade?  ");
     scanf(“%i”,&idade);
     puts("Qual é a sua altura?  ");
     scanf(“%f”,&altura);

     puts("\n Gravando seus dados no arquivo teste.txt\n");
      fprintf(arq,”%s %i %.2f \n”,nome,idade,altura);
     fclose(arq);

      printf("\nAgora vamos abrir o arquivo e ler os dados gravados”);
      arq = fopen(“teste.txt”, "r");
     if (arq==NULL)
     { printf("\nErro na abertura do arquivo teste.txt");
       getch();
        exit(1);
      }

     fscanf(arq,”%s %i %f \n”,nomearq,&idadearq,&alturaarq);
     printf(”\n Nome=%s Idade=%i Altura=%.2f \n”,
                nomearq,idadearq,alturaarq);
     fclose(arq);
      printf("\n\nFechando o arquivo. Fim do programa\n");
      puts("Hasta la vista, baby!\n");
     getch();
}
```

Exemplo 10.2

```
/* FILE2.c - Ilustra  o uso de feof */

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>

void main()
{
      FILE *arq;
     char nome[30];
     int idade, cont=0;
     float altura;

      arq = fopen("dados.dat", "w");
     if (arq == NULL)
     { printf("\nErro na criação do arquivo dados.dat");
       getch();
        exit(1);
      }

     for (;;)
     {  puts("Nome (. para finalizar):  ");
        scanf("%s",nome);
        if (strcmp(nome,".")==0)
           break;
        puts("Idade:  ");
        scanf("%i",&idade);
        puts("Altura:  ");
        scanf("%f",&altura);
        fprintf(arq,"%s %i %.2f \n",nome,idade,altura);
     }
     fclose(arq);

      printf("\nAgora vamos abrir o arquivo e ler os dados gravados");
      arq = fopen("dados.dat", "r");
     if (arq==NULL)
     { printf("\nErro na abertura do arquivo dados.dat");
       getch();
        exit(1);
      }

     while (!feof(arq))
     { fscanf(arq,"%s %i %f \n",nome,&idade,&altura);
       printf("\n Nome=%s Idade=%i Altura=%.2f \n",
                nome,idade,altura);
       ++cont;
     }
     fclose(arq);
      printf("\nArquivo com %d registros. Fechando o arquivo.\n",cont);
      puts("\nGoodbye!");
     getch();
}
```

10.4. As funções getc( ) e fgetc( )

- Arquivo de cabeçalho a ser incluído: **stdio.h**
- Estas funções são idênticas no modo de uso, a última sendo conservada para manter a compatibilidade com versões anteriores.
- Lêem caracteres de um arquivo aberto no modo leitura por **fopen()**.
- Recebem como argumento um ponteiro de arquivo.
- Devolvem um inteiro, mas o caracter está contido no byte menos significativo, o mais significativo sendo zero.
- Devolvem **EOF** ao ser atingido o final de arquivo.

Exemplo:

```
int ch;
FILE *fp;
do
{  ch = getc(fp);
} while(ch!=EOF);
```

10.5. As funções putc( ) e fputc( )

- Arquivo de cabeçalho a ser incluído: **stdio.h**
- Estas funções são idênticas no modo de uso, a última sendo conservada para manter a compatibilidade com versões anteriores.
-Escrevem caracteres em um arquivo que foi previamente aberto para escrita através da função **fopen()**.
- Recebem como argumento o caracter a ser escrito e um ponteiro de arquivo.

Exemplo:

```
int ch;
FILE *fp;
do
{    ch = getche();
     putc(ch,fp);
} while(ch!='.');
```

Exemplo 10.3:

```
/* FILE3.c - Ilustra a abertura e fechamento de arquivos,
            escrita e leitura nos mesmos */

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

void main()
{
      FILE *fp;
      char nome_arq[80];
      char ch;

      printf("Forneca o nome de um arquivo a ser criado\n");
      gets(nome_arq);

      printf("\nAgora vamos abrir o arquivo %s!\n", nome_arq);
      if ( (fp = fopen(nome_arq, "w") ) == NULL)
      { printf("Sinto muito: Erro na abertura do arquivo.\
               O programa sera abortado\n");
        exit(1);
      }

     puts("\nEntre com uma frase de efeito. Termine com um\
             ponto (.)\n");

      do
      {   ch = getche();
          putc(ch, fp);
      } while(ch!='.');

      printf("\n\nAgora, vamos fechar o arquivo %s\n", nome_arq);
      fclose(fp);

      printf("\n\nVamos abri-lo novamente, para que possamos ler\
             o que nele foi escrito\n");
      if ( (fp = fopen(nome_arq, "r") ) == NULL)
      { printf("Sinto muito: Erro na abertura do arquivo.\
              O programa sera abortado\n");
         exit(1);
      }

      puts("\nEis aqui a sua frase:\n");
      do
     { ch = getc(fp);
        putchar(ch);
      } while(ch!=EOF);

      printf("\n\nBem, fechemo-lo novamente\n");
      fclose(fp);
      puts("Ate mais ver!\n");
}
```

10.6. A Função "feof( )"

- Arquivo cabeçalho a ser incluído: **stdio.h**

- Determina quando o final de arquivo foi atingido na leitura de dados binários, mas também pode ser usada com arquivos texto: na leitura de um arquivo binário um valor inteiro igual ao de **EOF** pode ser lido, o que indicaria uma condição de fim de arquivo, apesar do final físico do mesmo não ter sido alcançado.

- Recebe como argumento um ponteiro para arquivo.

- Devolve verdadeiro se o final de arquivo tiver sido atingido e 0, caso contrário.

Exemplo:

```
while(!feof(fp))
    ch = getc(fp);
 /* Lê um arquivo binário até que o final de arquivo seja encontrado*/
```

10.7. A função `fputs( )`

- Arquivo cabeçalho a ser incluído: **stdio.h**

- Opera de forma semelhante a sua equivalente **puts().**

- Grava strings em um arquivo.

- Recebe como argumentos a string a ser escrita e um ponteiro para arquivo.

- Devolve **EOF** se ocorre um erro.

Exemplo:

```
char str[80] = "Alô, alô Terezinha.";
FILE *fp;
   …
fputs(str, fp);
    /*Escreve a string str no arquivo apontado por fp*/
```

10.8. A função `fgets()`

- Arquivo cabeçalho a ser incluído: **stdio.h**

- Opera de forma semelhante a sua equivalente **gets().**

- Lê strings de caracter de um arquivo.

- Recebe como argumentos o nome da string que vai receber o texto lido, o comprimento do texto a ser lido e um ponteiro para arquivo.

- A string é lida até que um caracter de nova linha seja lido ou que o comprimento especificado seja atingido. Um nulo é acrescentado ao final da string.

- Retorna um ponteiro para a string para onde a cadeia é direcionada ou um nulo, na ocorrência de erro.

Exemplo:

```
#define COMPRIMENTO 80
char str[COMPRIMENTO];
FILE *fp;
while(!feof(fp))
{ fgets(str, COMPRIMENTO, fp);
 }
```

Exemplo 10.4:

```
/* Fpufge.c: Ilustra o uso das funções fputs e fgets */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define COMPRIMENTO 80

void main()
{      FILE *fp1;
       FILE *fp2;
       char nome_arquivo1[COMPRIMENTO];
       char nome_arquivo2[COMPRIMENTO];
       char str[COMPRIMENTO];
       int conta_linha;

       puts("Forneca o nome de um arquivo a ser criado:");
       gets(nome_arquivo1);
       puts("Forneca outro nome de arquivo a ser criado:");
       gets(nome_arquivo2);

       if( (fp1=fopen(nome_arquivo1,"w")) == NULL)
      { printf("Problemas na abertura do arquivo %s\n",nome_arquivo1);
         printf("O programa sera abortado\n");
         exit(1);
       }
       if( (fp2=fopen(nome_arquivo2,"w")) == NULL)
      { printf("Problemas na abertura do arquivo %s\n",nome_arquivo2);
         printf("O programa sera abortado\n");
         exit(1);
       }

       puts("\nPrimeiramente vamos escrever algo no primeiro arquivo\n");
       do
      {  puts("Forneca uma frase. Tecle apenas <enter> para sair.");
          gets(str);
          strcat(str, "\n");      /* Acrescenta uma nova linha */
          fputs(str, fp1);
       } while(*str!='\n');

       puts("\nVamos agora fechar este arquivo para entao abri-lo novamente");
       puts("no modo de leitura somente\n");
       fclose(fp1);
       if( (fp1=fopen(nome_arquivo1,"r")) == NULL)
      {  printf("Problemas na abertura do arquivo %s\n",nome_arquivo1);
          printf("O programa sera abortado\n");
          exit(1);
       }
       puts("Agora, vamos ler o que foi escrito no primeiro arquivo");
       puts("e escrever o mesmo no segundo arquivo\n");

       conta_linha=0;
       while(!feof(fp1))
      {      conta_linha++;
             fgets(str, COMPRIMENTO, fp1);
             if (strcmp(str,"\n"))
            {       printf("Escrevendo no segundo arquivo:\n");
                    printf("Linha numero %d: %s\n",conta_linha, str);
                    fputs(str, fp2);
             }
       }

       puts("Fechando os dois arquivos\n");
       fclose(fp1);
       fclose(fp2);
       puts("Auf wiedersehen.");
}
```

10.9. A função `rewind( )`

- Arquivo cabeçalho a ser incluído: **stdio.h**
- Reposiciona o indicador de posição do arquivo no início do arquivo.
- Recebe por argumento um ponteiro para arquivo.

Exemplo 10.5:

```
/* Rewind.c: Ilustra o uso da função rewind() */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define COMPRIMENTO 80

void main()
{ FILE *fp1;
  char nome_arquivo1[COMPRIMENTO];
  char str[COMPRIMENTO];

  puts("Forneca o nome de um arquivo a ser criado:");
  gets(nome_arquivo1);

 printf("O arquivo sera aberto no modo de leitura-escrita\
         "w+\", para\n que possa ser lido apos rebobinado\n");
 if( (fp1=fopen(nome_arquivo1,"w+")) == NULL)
 { printf("Problemas na abertura do arquivo %s\n",nome_arquivo1);
   printf("O programa sera abortado\n");
   exit(1);
 }

 puts("\nEscrevendo algo no arquivo\n");
 do
 { puts("Forneca uma frase. Tecle apenas <enter> para sair.");
   gets(str);
   strcat(str, "\n");   /* Acrescenta uma nova linha */
   fputs(str, fp1);
 } while(*str!='\n');

 puts("Agora, vamos rebobinar o arquivo\n");
 rewind(fp1);

 puts("Agora, vamos ver o que foi escrito no arquivo:");

 while(!feof(fp1))
 { fgets(str, COMPRIMENTO, fp1);
   printf(str);
 }

 puts("Fechando o arquivo\n");
 fclose(fp1);
 puts("Hasta la vista, baby.");
}
```

10.10. A função `ferror( )`

- Arquivo cabeçalho a ser incluído: **stdio.h**
- Recebe como argumento um ponteiro para arquivo;
- Retorna verdadeiro na ocorrência de um erro durante a última operação no arquivo e falso, caso contrário. Deve ser chamada imediatamente após cada operação, porque a condição de erro é constantemente modificada.

Exemplo 10.6:

```
/* Ferror.c - Ilustra o uso da função ferror() */
#include<stdio.h>
#include<stdlib.h>

void main()
{     FILE *fp;
      char nome_arq[20];
      char str[50];

      puts("Forneca o nome de um arquivo a ser criado:\n");
      gets(nome_arq);

      puts("Abrindo o arquivo no modo de escrita");
      if( (fp=fopen(nome_arq,"w")) == NULL)
      {     puts("Nao foi possivel abrir o arquivo.");
            exit(1);
      }

      puts("Vamos escrever qualquer coisa no arquivo. Por ex:");
      puts("NONONONONONO");
      fprintf(fp,"NONONONONONO");

      puts("Esperamos que esta operacao nao resulte em erro.");
      puts("Afinal o arquivo foi mesmo aberto para escrita.");
      puts("Vamos testar a condicao de erro:");

      if (ferror(fp))
        printf("Erro de escrita no arquivo: %s\n", nome_arq);
      else
        puts("Nenhum erro ocorreu.");

      puts("Agora, vamos rebobinar o arquivo.\n");
      rewind(fp);

      puts("Agora, vamos ler o que foi escrito no arquivo.");
      fscanf(fp,"%s", str);

      puts("Vamos testar novamente a condicao de erro:");

      if (ferror(fp))
      { printf("Erro de leitura no arquivo: %s\n", nome_arq);
        puts("Isto era mesmo esperado, pois o arquivo foi aberto\
              apenas para escrita.\n");
      }
      else
        puts("Nenhum erro ocorreu.");

      puts("Au revoir!");
}
```

10.11. A função **`remove( )`**

- Arquivo cabeçalho a ser incluído: **stdio.h**
- Recebe como argumento o nome do arquivo a ser removido.
- Devolve zero se for bem sucedida e um valor diferente de zero, caso contrário.

Exemplo:

```
FILE *fp1;
char nome_arquivo[10] = "tmp.txt";
if ((fp1=fopen(nome_arquivo,"w")) == NULL)
     exit(1);
. . .
if(remove(nome_arquivo))
 {   printf("Arquivo nao pode ser apagado\n");
     exit(1);
 }
```

10.12. A função **`fflush( )`**

- Arquivo cabeçalho a ser incluído: **stdio.h**
- Esvazia o conteúdo de uma ***stream*** de saída.
- Recebe como argumento um ponteiro para arquivo. Se for chamada com um valor nulo, todos os arquivos abertos para saída são descarregados.
- Devolve 0 para indicar sucesso e **EOF**, caso contrário.

10.13. A função **`fwrite( )`**

- Arquivo cabeçalho a ser incluído: **stdio.h**
- Permite a escrita, em um arquivo, de dados maiores que 1 byte, para qualquer tipo de dado.
- Recebe como argumento um ponteiro para as informações que serão escritas no arquivo, o número de bytes a serem escritos, o número de itens a serem escritos, cada um com o número de bytes especificado, e um ponteiro para arquivo.
- Devolve o número de itens escritos, que será igual ao número especificado, a não ser na ocorrência de um erro.

Exemplo:

```
FILE *fp;
int num = 10;
. . .
fwrite(&num,sizeof(int),1,fp);
/* Escreve o conteúdo de num uma vez
   no arquivo apontado por fp */
```

10.14. A função `fread( )`

- Arquivo cabeçalho a ser incluído: **stdio.h**

- Permite a leitura, em arquivo, de dados, maiores que 1 byte, para qualquer tipo de dado.

- Recebe como argumento um ponteiro para uma região de memória que receberá os dados do arquivo, o número de bytes a serem lidos, o número de itens a serem lidos, cada um com o número de bytes especificado, e um ponteiro para arquivo.

- Devolve o número de itens lidos, o qual pode ser menor que o número especificado, se o final do arquivo for atingido ou na ocorrência de um erro.

Exemplo:

```
FILE *fp;
double db;
. . .
fread(&db,sizeof(double),1,fp);
 /* Le um double do arquivo apontado por fp */
```

Exemplo 10.7:

```
/* FreFwri.c : Ilustra o uso das funções fread(), fwrite() e remove() */

#include<stdio.h>
#include<stdlib.h>

void main()
{     FILE *fp;
      double db=11.1213, db1;
      int i=5, i1;
      char nome_arq[20];

      puts("Forneca o nome do arquivo a ser criado:");
      scanf("%s",nome_arq);
      puts("\nAbrindo o arquivo no modo binario leitura/escrita:");
      if( (fp=fopen(nome_arq, "wb+"))== NULL)
      { puts("Voce nao esta com sorte: o arquivo nao pode ser aberto!");
        puts("O programa sera abortado.");
        exit(1);
      }
      printf("\nEscrevendo com fwrite() no arquivo %s o conteudo de db=%f e\
              i=%d\n", nome_arq, db, i);
      fwrite(&db, sizeof(double), 1, fp);
      fwrite(&i , sizeof(int)   , 1, fp);
      puts("\nRebobinando o arquivo.");
      rewind(fp);
      printf("\nLendo com fread() do arquivo %s para as variaveis db1\
             e i1\n", nome_arq);
      fread(&db1, sizeof(double), 1, fp);
      fread(&i1 , sizeof(int)   , 1, fp);
      printf("\nOs valores lidos sao: db1 = %f e i1 = %d\n",db1, i1);
      puts("\nFechando o arquivo...");
      fclose(fp);

      printf("\nDeletando o arquivo %s\n",nome_arq);
      if(remove(nome_arq))
      {  printf("O arquivo %s nao pode ser apagado!\n",nome_arq);
         exit(1);
      }
}
```

10.15. A Função `fprintf( )`

- Arquivo cabeçalho a ser incluído: **stdio.h**
- Comporta-se da mesma forma que **printf()**, operando, entretanto, com arquivos.
- Seu primeiro argumento é um ponteiro para arquivo, sendo os demais argumentos iguais aos de **printf()**.

10.16. A função `fscanf( )`

- Arquivo cabeçalho a ser incluído: **stdio.h**
- Comporta-se da mesma forma que **scanf()**, exceto por operar com arquivos.
- Seu primeiro argumento é um ponteiro para arquivo, sendo os demais argumentos como os de **scanf()**.

Exemplo 10.8:

```
/* FprFsc.c: Ilustra o uso das funções fprintf() e
            fscanf() para gravação e leitura de arquivos */
#include<stdio.h>
#include<stdlib.h>

void main()
{       FILE *fp;
        float db=10.1010, db1;
        int i=10, i1;
        char nome_arq[20];

        puts("Forneca o nome do arquivo a ser criado:");
        scanf("%s",nome_arq);
        puts("\nAbrindo o arquivo no modo escrita:");
        if( (fp=fopen(nome_arq, "w"))== NULL)
      { puts("O arquivo nao pode ser aberto!");
          puts("O programa sera abortado.");
          exit(1);
        }

        printf("\nEscrevendo com fprintf() no arquivo %s o conteudo\
               de db=%f e i=%d\n", nome_arq, db, i);
        fprintf(fp, "%f %d", db, i);
        puts("Fechando o arquivo.");
        fclose(fp);
        puts("Abrindo-o agora no modo de leitura.");
        if( (fp=fopen(nome_arq, "r"))== NULL)
      {  puts("O arquivo nao pode ser aberto!");
           puts("O programa sera abortado.");
           exit(1);
        }

        printf("\nLendo com fscanf() do arquivo %s para as variaveis\
              db1 e i1\n", nome_arq);
        fscanf(fp,"%f %d", &db1, &i1);
        printf("\nOs valores lidos sao: db1 = %f e i1 = %d\n",db1, i1);
        puts("\nFechando o arquivo...");
        fclose(fp);
}
```

10.17. A função `fseek( )`

- Arquivo cabeçalho a ser incluído: **stdio.h**

- Recebe como argumentos um ponteiro para arquivo, um valor inteiro, que deve ser uma das macros definidas em **stdio.h**:

SEEK_SET Início do arquivo
SEEK_CUR Posição atual
SEEK_END Final do arquivo

e o número de bytes a partir da posição dada pela macro usada.

- Modifica o indicador de posição de arquivo, movendo-o o número de bytes especificado a partir da posição especificada.

- Retorna 0 se bem sucedida e um valor não nulo quando da ocorrência de erro.

Exemplo 10.9:

```
/* Fseek.c: Mostra o uso da funcao fseek() */
#include<stdio.h>
#include<stdlib.h>

void main()
{       FILE *fp;
        char nome_arq[20];
        char ch;
        long int numbytes;
        int i;
        puts("Primeiramente, vamos construir um arquivo,\
               contendo uma sequencia de chars.\n");
        puts("Forneca o nome do arquivo:");
        gets(nome_arq);

        if( (fp = fopen(nome_arq,"w")) == NULL )
      { printf("Sinto nos ossos, mas o arquivo nao pode ser aberto\n");
                printf("Ate a proxima\n");
                exit(1);
        }
        puts("Escrevendo os caracteres de \'a\' a \'z\' no arquivo.");
        for(ch=97; ch<=122; ch++)
                fprintf(fp,"%c", ch);
        puts("Fechando o arquivo para reabri-lo no modo de leitura.");
        fclose(fp);

        puts("Reabrindo o arquivo.");
        if( (fp = fopen(nome_arq,"r")) == NULL )
      {
                printf("Sinto nos ossos, mas o arquivo nao pode ser aberto\n");
                printf("Ate a proxima\n");
                exit(1);
        }
        puts("Posicionando o arquivo de byte em byte desde o inicio do mesmo.");
        for(numbytes=0L; numbytes<=25L; numbytes++)
      {      if( fseek(fp,numbytes,SEEK_SET) )
             {  printf("Erro na busca\n");
                        exit(1);
                }
                printf(" O byte em %ld e %c\n", numbytes, getc(fp));
        }
        fclose(fp);
}
```

10.18. As *Streams* Padrão

- Toda vez que um programa C é executado são abertas três ***streams***:
  - **stdin:** entrada padrão;
  - **stdout**: saída padrão;
  - **stderr:** saída de erro padrão.

- Estas ***streams*** são em geral usadas para ler a partir do teclado (a entrada padrão) e para escrever na tela (saída padrão e saída de erro padrão).

- São como ponteiros de arquivos e qualquer função que tenha como argumento uma variável do tipo **FILE ***, pode usá-las no lugar desta variável.

Exemplos:

```
fprintf(stdin, “%d\n”, i);
fscanf(stdout,”%d”, &I);
fputs(“Como vai?”, stdin);
```

- Uma vez que o DOS suporta redirecionamento de E/S, é possível redirecionar a saída e a entrada padrão para arquivos.

Exemplo:

```
PROG < INPUT.TXT > OUTPUT.TXT
```

Se PROG é o nome de um executável, esta é a maneira de direcionar a entrada padrão do programa PROG para o arquivo INPUT.TXT e a saída padrão para o arquivo OUTPUT.TXT.

10.19. Enviar saída para a impressora com fprintf

- C oferece várias funções de biblioteca que podem ser utilizadas para enviar dados a vários destinos.

- Em complemento, o arquivo de cabeçalho <stdio.h> define nomes que representam, além das streams padrão, a impressora como destino de saída:

  - **Stdprn**: saída para impressora padrão (dispositivo identificado como PRN no DOS).

- A função printf sempre envia sua saída para o stdout.

- Mas outras funções permitem especificar os destinos dos dados de saída, como a função **fprintf( )** da biblioteca <stdio.h >.

- **`fprintf`** pode ser usada para enviar dados à impressora.

Sintaxe: `fprintf (destino, "formato", [lista de variáveis]);`

- Destino especifica para onde a saída será enviada. Formato e valor trabalham como com printf .

- Se o destino for **stdout** fprintf se comporta como printf.

- Se o destino for **stdprn** fprintf envia a saída para a impressora.

- Para avançar uma página na hora da impressão, deve-se enviar um caracter de avanço de página \f ("Form feed").

Exemplo 10.10:

```
/* .......................................
      Fprintf.c: Ilustra o uso da funcoes fprintf() para envio de dados
      para uma impressora. Este programa pode não funcionar em alguns pcs...
..........................................*/
#include<stdio.h>
#include<stdlib.h>
#include <conio.h>
void main()
{
      float db=10.1010, db1;
      int i=10, i1;
      char nome[20]="Teste de impressora";

      clrscr();

      puts("Enviando dados para a impressora:");

      printf("\nEscrevendo com fprintf() na impressora o conteudo de\
                  \n frase=%s db=%f e i=%d\n",  nome, db, i);

      fprintf(stdprn, " %s %f %d", nome, db, i);

      printf("\n\n");
      puts("Finaliza a impressao e o programa.");
      getch();
}
```

Lista de Exercícios Nº 10

1. *Substitua, no exemplo 10.3 o uso da função putc pela função fprintf.*

2. *Modifique o programa do exemplo 10.4 de forma que não seja necessário abrir o arquivo apontado por *fp1 duas vezes.*

3. *Crie um arquivo contendo qualquer texto, usando um editor do DOS ou do Windows. Escreva um programa que o abra no modo apend ("a") e permita ao usuário que leia o que o arquivo contém e, em seguida, escreva no mesmo arquivo o que quiser. Dê ao usuário a opção de deletar ou não o arquivo antes de sair do programa.*

4. *Fazer um programa que leia um arquivo-texto contendo os nomes de várias pessoas e gere um arquivo-texto de saída com os tamanhos dos nomes lidos no arquivo de entrada.*

5. *Fazer um programa para calcular a folha de pagamentos dos funcionários de uma pequena empresa. Devem ser lidos, de um arquivo, os seguintes dados de cada funcionário: nome, número de horas trabalhadas, número de horas-extra e número de dependentes. O salário é calculado da seguinte maneira:*

   *SALÁRIO BRUTO = (número de horas trabalhadas * VHT) +*
   *número de horas-extra * VHE) +*
   *(número de dependentes * VCD)*

   SALÁRIO LÍQUIDO = SALÁRIO BRUTO - (desconto IR) - (desconto INSS)

   Os valores de VHT, VHE, VCD, desconto de IR e desconto de INSS devem ser lidos de um arquivo, como no exemplo mostrado abaixo.

VHT	VHE	VCD	desconto IR	desconto INSS
R$ 14.00	R$ 21.00	R$ 100,00	12% do SALÁRIO BRUTO	8% do SALÁRIO BRUTO

6. *O programa deve imprimir na tela e num arquivo de saída um cabeçalho com o nome da empresa e os nomes de todos os funcionários seguidos do seu salário bruto, do desconto do IR, do desconto do INSS e do seu salário líquido.*

7. *Fazer um programa que leia um arquivo, de no máximo 100 registros, contendo os nomes e os preços de vários produtos e armazene-os numa tabela (vetor de registros). Em seguida, pergunte ao usuário o nome do produto (flag='FIM') e forneça o preço correspondente.*

8. *Fazer um programa que leia um arquivo, de no máximo 100 registros, contendo os nomes e as estaturas de várias pessoas. Em seguida, imprima os dados do arquivo na tela e o nome e a estatura da pessoa mais alta.*

11. Ponteiros

11.1. Armazenamento de dados na memória

- Cada espaço da memória tem um endereço associado a ele.

2A00:	
2A01:	
2A02:	
2A03:	
2A04:	
2A05:	

- Ao se declarar uma variável, o compilador reserva um espaço de memória para poder armazenar o valor desta variável, e quando se deseja saber este valor, o programa procura no endereço de memória da variável.

Exemplo:

```
int dado; /* O compilador associa "dado com" um endereço
             (2A04, por exemplo)*/

dado = 10;  // O valor 10 é armazenado no endereço de dado (2A04)

printf("\n%d",dado); /* O valor armazenado no endereço de dado
                        é impresso */
```

- Poder-se-ia criar uma variável que guardasse o endereço de *dado*, por exemplo *apdado*. Assim, o valor de *apdado* seria, no caso do exemplo, 2A04. Ou seja, a variável *apdado* indica a localização de *dado* na memória.

- As variáveis do tipo *apdado* são chamadas de ponteiros.

11.2. Declaração de ponteiros: operadores & e *

- Antes de ser usado, um ponteiro deve ser declarado, como qualquer variável em C.
- O nome de um ponteiro segue as mesmas regras para nomes de variáveis.

Sintaxe:
```
tipo *nome_do_ponteiro;
```

tipo é o tipo da variável para a qual o ponteiro está apontando.

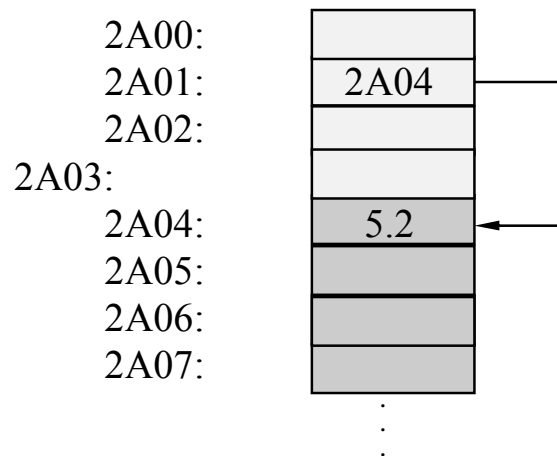
Eemplos:
```
float *apnota, *apvalor;
long resto, *apresto;
 int  dado, *apdado;
```

- **O operador &:**

- Após ser declarado, um ponteiro deve ser inicializado para que aponte para um valor útil. Para isto deve-se utilizar o operador **&**.

- O operador unário **&**, chamado **endereço de**, fornece o endereço do seu operando.

Exemplos:
```
float nota = 5.2, *apnota;
apnota=&nota;     /* Atribui a apnota o endereço de memória que
                     contém a variável nota: diz-se que
                     apnota aponta para nota */
```

Endereço	Conteúdo
2A00:	
2A01:	2A04
2A02:	
2A03:	
2A04:	5.2
2A05:	
2A06:	
2A07:	
. . .	

- Os ponteiros apontam para o byte de posição mais baixa de uma variável.

- O operador **&** só se aplica a variáveis e elementos de vetor.

- O operador **&** não pode ser aplicado a expressões, constantes ou variáveis do tipo **register**.

O operador *:

- É um operador unário chamado de operador de **indireção** ou de **deferenciação**.
- Quando aplicado a um apontador, acessa o objeto para o qual o apontador aponta, ou seja, refere-se ao valor armazenado na posição de memória da variável para a qual o ponteiro está apontando.
- Pode-se acessar o conteúdo de uma variável usando o seu nome (acesso direto), ou usando ponteiros (acesso indireto ou indireção).

Exemplo:

```
float dado=15, *apdado;
apdado=&dado;
printf("%d", dado);      // Saída: 15
printf("%d", apdado);    // Saída: 2A04
printf("%d", *apdado);   // Saída: 15
```

- Se um ponteiro *px aponta para uma variável x, então *px pode aparecer em qualquer contexto em que a variável x apareceria.

Exemplo:

```
int y, x = 2, *px;
px = &x;
*px += 5;          // Incrementa x de 5
++(*px);           // Incrementa x de 1
--*px;             // Incrementa x de 1
y = *px+2;         /* Atribui a y o resultado
                      da soma de x com 2 */
```

Exemplo 11.1:

```
/* Ponteiro.c: Exemplifica acesso direto e indireção*/

#include <stdio.h>
#include <conio.h>

int dado=2, *apdado;

main( )
{
        puts("Vamos fazer apdado apontar para dado\n");
        apdado=&dado;
        printf("\nValor de dado=  %d ", dado);
        printf("\Valor de apdado= %p: ", apdado);
        printf("\nValor de *apdado: %d ", *apdado);
        printf("\nValor de &dado: %p ", &dado);
        getch();

        return 0;
}
```

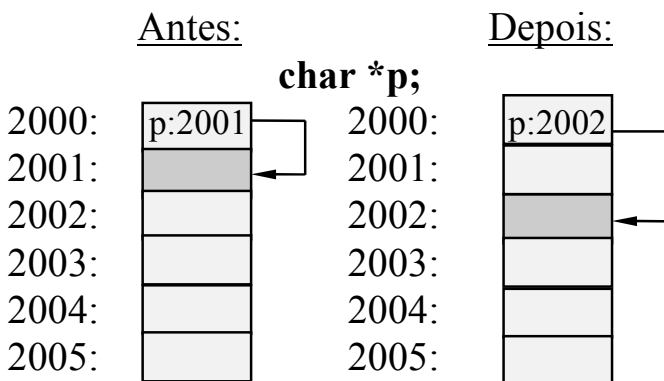
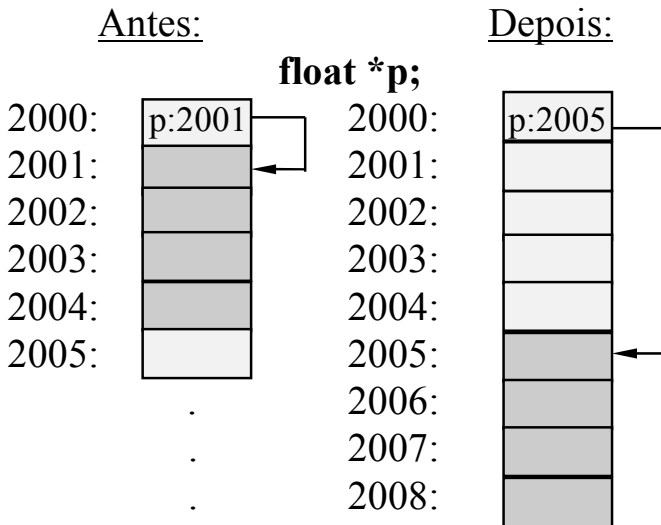
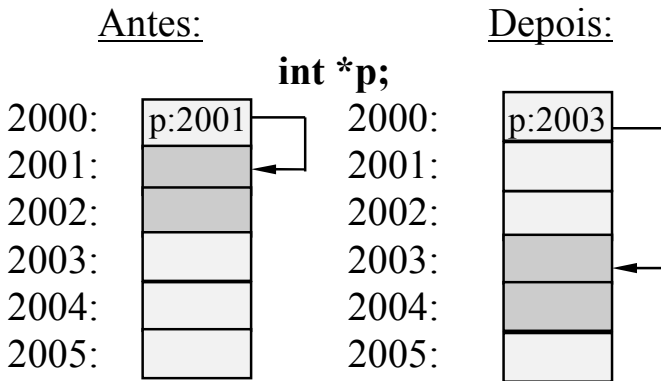
11.3. Aritmética de Ponteiros: Soma e Subtração de Inteiros

- As únicas operações possíveis entre ponteiros e inteiros são a subtração e adição.

11.3.1. Adição de inteiros a ponteiros

- A adição de um inteiro a um ponteiro faz com que o mesmo aponte para o endereço resultante da soma do endereço atual com o resultado da multiplicação do inteiro em questão pelo tamanho em bytes do tipo de objeto para o qual ele aponta.

Exemplo:

```
/* Suponhamos que p seja um ponteiro que esteja apontando para o endereço 2001 */

p++;

/* Se p aponta para um caracter:
     p++ irá apontar para 2001 + 1*1 byte = 2002
   Se p aponta para um inteiro:
     p++ irá apontar para 2001 + 1*2 bytes = 2003
   Se p aponta para um float:
     p++ irá apontar para 2001 + 1*4 bytes = 2005 */
```

- Pode-se somar qualquer inteiro a um ponteiro.

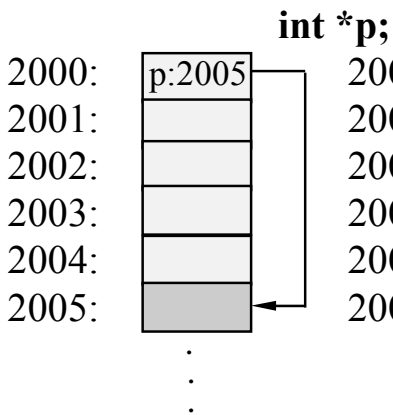
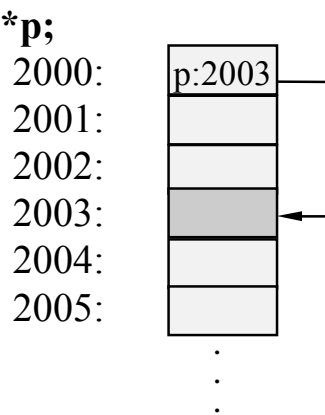
Exemplo:

```
int x = 2, *px;
px = &x ;
px = px + 5; /* px passa a apontar para o quinto
                elemento do tipo int após o elemento
                para o qual ele atualmente aponta */
```

11.3.2. Subtração de inteiros a ponteiros

- A subtração de um inteiro de um ponteiro, faz com que o mesmo aponte para o endereço resultante da subtração do endereço atual do resultado da multiplicação do inteiro em questão pelo tamanho em bytes do tipo de objeto para o qual ele aponta.

Exemplo:

```
     /* Suponhamos que p seja um ponteiro que esteja apontando para
o endereço 2005 */

     p--;

     /* Se p aponta para um caracter:
          p-- irá apontar para 2005 - 1*1 byte = 2004
        Se p aponta para um inteiro:
          p-- irá apontar para 2005 - 1*2 bytes = 2003
        Se p aponta para um float:
          p-- irá apontar para 2005 - 1*4 bytes = 2001 */
```

- Pode-se subtrair qualquer inteiro de um ponteiro.

Exemplo:

```
        float x = 2.3, *px;
        px = &x ;
        px = px - 3;
                 /* px passa a apontar para o terceiro
                    elemento do tipo float anterior ao elemento
                    para o qual ele atualmente aponta */
```

Exemplo 11.2:

```
/* PontArit.c: Este programa ilustra a aritmetica de ponteiros */

#include <stdio.h>

void main()
{
      int i = 2, *pi;
      char c = 'a', *pc;
      float f = 23.57, *pf;

      pi = &i;
      pc = &c;
      pf = &f;
      printf("pi aponta para o endereco: %d\n", pi);
      printf("pc aponta para o endereco: %d\n", pc);
      printf("pf aponta para o endereco: %d\n", pf);

      puts("Somando 4 a todos os ponteiros.\n");

      pi = pi + 4;
      pc = pc + 4;
      pf = pf + 4;
      printf("Agora, pi aponta para o endereco: %d\n", pi);
      printf("Agora, pc aponta para o endereco: %d\n", pc);
      printf("Agora, pf aponta para o endereco: %d\n", pf);

      puts("\nSubtraindo 5 de todos os ponteiros.\n");

      pi = pi - 5;
      pc = pc - 5;
      pf = pf - 5;
      printf("Agora, pi aponta para o endereco: %d\n", pi);
      printf("Agora, pc aponta para o endereco: %d\n", pc);
      printf("Agora, pf aponta para o endereco: %d\n", pf);
}
```

11.4. Atribuição de Ponteiros

- Um ponteiro pode ser atribuído a outro ponteiro.

Exemplo:

```
int x = 2;
int *p1x, *p2x;
p1x = &x;
p2x = p1x;  // Atribui o conteudo de p1x para p2x
```

- O resultado desta operação é que o ponteiro do lado esquerdo passa a apontar para o mesmo endereço para o qual aponta o ponteiro do lado direito do sinal de atribuição.

11.5. Ponteiros e Vetores/Matrizes/Strings/Structs

- Há uma forte relação entre ponteiros e vetores/matrizes:

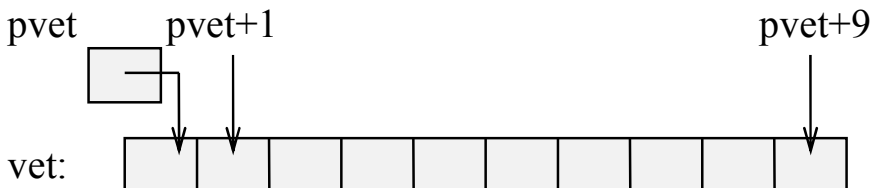
- O nome de uma matriz/vetor sem os colchetes é um ponteiro para o primeiro elemento desta matriz/vetor.

  Exemplo:

```
int vet[10], *pvet;
pvet = vet;  /* Faz com que pvet aponte para o
             primeiro elemento do vetor vet*/
pvet = &vet[0]  // Faz o mesmo trabalho
```

- Há dois métodos para acessar elementos de matrizes/vetores: aritmética de ponteiros ou indexação da matriz/vetor.

  Exemplo:

```
int elemento4, vet[10], *pvet;
pvet = vet;
elemento4 = vet[3]; /* Acessa o quarto elemento
                       do vetor vet */
elemento4 = *(pvet+3); /* Faz o mesmo */
```

Exemplo 11.3:

```
/* PontVet.c - Mostra como acessar os elementos de um vetor usando ponteiros. */

#include <stdio.h>

int vet[8];
int *pvet;
int cont;

void main( )
{     printf("\nEntre com oito números inteiros.\n");
      for(cont=0;cont<8;cont++)
      scanf("\n%d",&vet[cont]);

      printf("\nUsando incremento de ponteiros para imprimir vet:\n");

      pvet=vet;

      for(cont=0;cont<8;cont++)
         printf("\n vet[%d] = %d",cont,*pvet+cont);
}
```

- Como strings nada mais são do que vetores de caracteres, podem-se declará-las como ponteiros para caracter.

Exemplo:

```
char cadeia1[9] = "Bom dia!";
char *cadeia1   =  "Bom dia!"; /* As duas
         declarações são equivalentes */
```

Exemplo 11.4:

```
/* PontStr.c: Ilustra o uso de cadeias de caracteres (strings) como ponteiros */

#include <stdio.h>

void main()
{     int i;

      char cadeia1[100]= "Ha homens que, com falsa superioridade,\nzombam das
tarefas humildes\n";
      char *cadeia2 = "como se nao fossem imprescindiveis ao exito \ndos trabalhos
de maior envergadura.\n";

      puts("Reflexoes sobre a importancia das coisas pequenas:\n");

      for(i=0; cadeia1[i]!='\0'; i++)
            printf("%c",cadeia1[i]);

      for(;*cadeia2 != '\0';cadeia2++)
            printf("%c",*cadeia2);
 }
```

-Os ponteiros para uma estrutura funcionam como os ponteiros para qualquer outro tipo de dados.

- Declaração de ponteiros para estruturas

```
      struct Nome_Estrutura {...} *NomePonteiro;
```
ou
```
      Tipo_Estrutura  *NomePonteiro;
```

Exemplos:
```
  struct Pessoa { char nome[40];
                  int idade; } *ppessoa;

  typedef struct {  char nome[40];
                    int idade; } PESSOA;
  PESSOA *ppessoa;
```

-Para acessar um elemento da estrutura através de um ponteiro usa-se o operador seta ' ->'

NomePonteiro->Elemento

Exemplo 11.5:

```
//PontStru.c: ilustra o uso de ponteiros para estruturas
#include <stdio.h>
#include <conio.h>
#include <string.h>
typedef struct { char nome[40];
                 int idade; } PESSOA;
void main ( )
{     PESSOA  a, b;
      PESSOA  *ppessoa;
      ppessoa = &a;

   clrscr ( );
   printf ("Informe o nome da pessoa: ");
   gets (ppessoa->nome);
   printf ("Informe a idade da pessoa: ");
   scanf ("%d", &ppessoa->idade);
   strcpy (b.nome, ppessoa->nome);
   b.idade = ppessoa->idade;
   printf ("\nNome da pessoa: b.nome= %s ", b.nome);
   printf ("\nIdade da pessoa: a.idade= %d", a.idade);
   getch();
}
```

11.6. Aritmética de Ponteiros: Subtração entre Ponteiros

- A operação de subtração tem sentido se estiverem envolvidos dois ponteiros que apontam para o mesmo tipo de objeto.
- O resultado desta operação é um valor inteiro que é igual ao número de objetos do tipo para o qual os ponteiros apontam que se interpõem entre os dois ponteiros.

Exemplo:

```
int vet[10];
int cont;
cont= &vet[6]-&vet[0];     /* Resulta em 6, pois vet[6]
                              está em um endereço
                               mais alto que vet[0] */
cont= &vet[0]-&vet[6];     /* Resulta em -6 pois vet[0]
                               está em um endereço
                               mais baixo que vet[6] */
```

- Esta é a chamada operação de diferenciação.

11.7. Comparação entre Ponteiros

- Quando dois ponteiros apontam para um objeto comum como, por exemplo, uma matriz pode-se compará-los usando os operadores relacionais = =, !=, >, <, => e >=.

Exemplo:

```
float vet[5];
int e_maior;
e_maior=(&vet[5] > &vet[3]);

/* Atribui 1 a e_maior, ja que &vet[5] aponta para uma
  posição de memória mais alta que &vet[3] */
```

Exemplo 11.6:

```
/* PontSubt.c: Ilustra a subtracao e a comparacao entre ponteiros */

#include <stdio.h>
#include <conio.h>

void main()
{
        double vet[5];
        int i, dif;
        double *pvet;

        puts("Forneca 5 numeros para um vetor double:\n");
        for(i=0; i<5; i++)
        {       printf("\n vet[%d]:",i);
                scanf("%f",&vet[i]);
        }

        pvet = vet;

        puts("\nVamos examinar a diferenca entre os enderecos destes elementos.\n");
        puts("Pressione qualquer tecla para continuar\n");
        getch();
        for(i=1; i<5; i++)
        {       if(&vet[i]>pvet)
                   printf("\n &vet[%d]=%d \"maior\" que pvet=%d \n",i,&vet[i],pvet);
                else
                   printf("\n &vet[%d]=%d \"menor\" que pvet=%d\n",i,&vet[i],pvet);

                printf("\n &vet[%d]-vet=%d\n",i,&vet[i]-vet);
        }

}
```

11.8. Matrizes/Vetores de Ponteiros

- Da mesma forma como se podem declarar matrizes/vetores de qualquer tipo de variável, também se podem declarar matrizes/vetores de ponteiros para qualquer tipo de variável.

Exemplo:

```
        int i=5, j, *vet[5];

/* Declara dois inteiros e um vetor de ponteiros para inteiros */
/* vet (sem índices) é um ponteiro para o vetor de ponteiros para inteiros */

        vet[1] = &i;
/* Atribui o endereço de uma variável inteira para o primeiro elemento do vetor vet */

        j = *vet[1];
/* Atribui o valor da variável para a qual vet[1] aponta para j */
```

Exemplo 11.7:

```
/* PontdeVet.c: Ilustra o uso de vetores de ponteiros */

#include<stdio.h>

void main()
{
        int i;
        char *poema1[]=
      { "        HIPOTESE",
                "E se Deus é canhoto",
                "e criou com a mao esquerda?",
                "Isso explica, talvez, as coisas deste mundo.",
                "                     Carlos Drummond de Andrade"
        };

        char *poema2[]=
        {"A vida é a arte do encontro,\n",
         "embora haja tantos desencontros pela vida...\n"
         "                    Vinicius de Moraes"
        };

        puts("Um pequeno poema:\n");
        for(i=0; i<5; i++)
                puts(poema1[i]);

        puts("\nAgora outro:\n");
        for(i=0; i<2; i++)
     {
                while(*poema2[i]!='\0')
                        printf("%c",*(poema2[i]++));
        }
}
```

11.9. Ponteiros como Argumentos de Funções

- Há duas maneiras de se passar argumentos para funções:

- <u>Chamada por valor:</u> Esta é a maneira usual.

-O valor é copiado do argumento para o parâmetro formal da função.

- Quaisquer alterações feitas a estes parâmetros dentro da função não irão afetar as variáveis usadas como argumentos para chamá-la.

- <u>Chamada por referência:</u>

- O que é copiado no parâmetro formal é o endereço da variável.

- Dentro da subrotina o argumento real utilizado na chamada é acessado através do seu endereço, podendo ser, desta forma, alterado.

Exemplo:

```
void func(int *i,int *j);
```

```
/* Declaração de um protótipo de função que recebe valores por referência */
```

```
int i1, i2;
func(&i1, &i2);
```

```
/* Chamada à função, passando valores por referência */
```

Exemplo 11.8:

```
/* PontRefer.c: Ilustra a passagem de argumentos por referência em funções */

#include<stdio.h>

void troca1(int m, int n);
void troca2(int *m, int *n);

void main()
{int i1, i2;
printf("Forneca o valor de dois inteiros\n");
scanf("%d %d", &i1, &i2);

printf("Temos i1=%d e i2=%d\n", i1,i2);
printf("\nVamos troca-los de lugar usando a funcao troca1, passando-os por valor a funcao.\n");
troca1(i1,i2);

printf("Depois da troca, de volta a funcao main, temos i1=%d e i2=%d\n", i1,i2);
printf("\nVamos troca-los de lugar usando a funcao troca2, passando-os por referencia a funcao.\n");
troca2(&i1,&i2);

printf("Depois da troca, de volta a funcao main, temos i1=%d e i2=%d\n", i1,i2);
}

void troca1(int i1, int i2)
{int tmp;
tmp = i1;
i1 = i2;
i2 = tmp;
printf("Depois da troca, dentro da funcao troca1, temos i1=%d e i2=%d\n", i1,i2);
}

void troca2(int *i1, int *i2)
{int tmp;
tmp = *i1;
*i1 = *i2;
*i2 = tmp;
printf("Depois da troca, dentro da funcao troca1, temos i1=%d e i2=%d\n", *i1,*i2);
}
```

Lista de Exercícios 11

__1.__ Modifique o valor da variável __dado__ do exemplo __11.1__, somando-o a um valor inteiro fornecido pelo usuário, porém acessando-o através do ponteiro que aponta para ele.

__2.__ Defina um vetor de inteiros de comprimento 12. Crie um ponteiro que aponta para ele. Preencha o vetor com valores iguais a de seus índices (Ex: a[2] = 2), acessando os seus elementos através do ponteiro. Forneça ao usuário a opção de ver os elementos do vetor das seguintes formas:

- *Ver todos os elementos um a um;*
- *Ver apenas os elementos de índices pares;*
- *Ver apenas os elementos cujos índices são divisíveis por 3;*
- *Ver apenas os elementos cujos índices são divisíveis por 4;*

Use aritmética de ponteiros.

__3.__ Defina um vetor de qualquer tipo com 10 elementos. Peça ao usuário que forneça dois inteiros entre 0 e 9. Apresente-lhe uma mensagem dizendo qual o elemento que se localiza na posição de memória mais alta:

- *Usando subtração de ponteiros.*
- *Usando comparação entre ponteiros.*

__4.__ Escreva uma função que recebe um valor inteiro entre 1 e 5 fornecido pelo usuário na função principal e imprime uma mensagem conforme o valor recebido, do tipo "Você escolheu a opção 1", se o valor for 1. Use para tanto um vetor de ponteiros para strings.

__5.__ Escreva um programa que recebe três números diferentes, os atribui a três variáveis __a__, __b__ e __c__ e chama uma função que coloca em __a__ o valor do menor, em __b__ o do intermediário e em __c__ o do maior e que ao retornar para a função principal imprima estes números. Use chamada por referência.

12. Pré-Processador

- Programa que executa antes do compilador.
- Prepara o código fonte para ser compilado.
- Todas as diretivas do pré-processador C começam com o símbolo #.

São elas:

```
#include
#define
#if
#ifdef
#ifndef
#else
#elif
#endif
#undef
#error
#line
#pragma
```

- As duas primeiras são as mais usadas.

12.1. A diretiva #include

Instrui o compilador a ler outro arquivo fonte e adicioná-lo àquele que contém a diretiva.

Exemplo:

```
#include <stdio.h>
#include "meucab.h"
```

- Quando o nome do arquivo está entre aspas, este será procurado no diretório de trabalho corrente.

- Quando o mesmo se encontra entre os sinais de maior e menor, será procurado no diretório especialmente criado para arquivos de inclusão.

12.2. A diretiva #define

- Define um identificador e uma cadeia que o substituirá em todo código fonte, processo este denominado por substituição de macro.

Exemplos:

```
#define imprimir printf
#define  E_MS "Erro padrão na entrada.\n"
. . .
imprimir(E_MS);

#define TAMANHO_MAX 100
. . .
float nome[TAMANHO_MAX];
for( i = 0; i < TAMANHO_MAX; i++)
    printf( "Nome = %f\n", nome[i]);
```

12.2.1. Macros de funções

- O nome da macro pode ter argumentos associados, o que permite a definição de macros de funções.

- Os argumentos não precisam ter um tipo especificado a priori.

Eemplos:

```
#define metade(valor)  ((valor)/2)

#define soma(a,b,c,d)  ((a)+(b)+(c)+(d))

#define ABS(a) (a)<0 ? -(a) : (a)
. . ..
printf("O valor absoluto de -1 é: %d\n",ABS(-1));

#define MAX(A,B) ((A) > (B) ? (A) : (B))
. . .
printf("O maior entre os números 3 e 5 é: %d\n",
               MAX(3,5));
```

- Todos os parâmetros da lista devem ser usados no string de substituição, não sendo então possível uma definição como a seguinte:

```
#define soma(a,b,c,d)  ((a)+(b)+(c))
```

- Os parênteses devem ser colocados logo após o nome definido (por exemplo soma( )), não podendo haver qualquer espaço vazio entre eles.

- O uso de parênteses no string de substituição evita que se obtenha resultados inesperados.

Exemplo:

```
#define quadrado(x)  x*x

resultado=quadrado(a+b); // resultado = a+b*a+b = a+(b*a)+b

#define quadrado(x)  (x)*(x)

resultado=quadrado(a+b); // resultado=(a+b)*(a+b)
```

- Se for usado o operador de literal alfanumérico (#) antes de uma string de substituição, o argumento será convertido em um string entre aspas, com o parâmetro substituído pelo string real.

Exemplo:

```
#define imprime(x)  printf(#x)
    . . .

imprime(roteiro);            // printf("roteiro")

#define dprint(expr) printf(#expr " = %f\n", expr)
    . . .

dprint(x/y);                 //   printf("x/y = %f\n", x/y)
```

- O operador ## concatena duas palavras.

Exemplo:

```
#define concat(a,b)  a ## b
#define COMECO   "Ola,"
#define FIM      "como vai?"
...
printf(concat(COMECO,FIM));
```

Exemplo 12.1:

```
/* Define.c: Ilustra o uso da diretiva #define */

#include <stdio.h>

#define MENSAGEM "Vamos mostrar alguns usos da diretiva define.\n"
#define concatena(a,b) a##b
#define imprimir printf
#define ler     scanf
#define RAZAO(a,b)  (a)/(b)
#define DEZ    10
#define ImprimindoMensagem(mens)   imprimir(#mens);
#define ImprimindoRazao(razao,x,y) printf(#razao" = %f",
            RAZAO(x,y));
#define PulaLinha           printf("\n");

void main()
{
     float a,b;

     imprimir(concatena(MEN,SAGEM));
     imprimir("Forneca dois floats: a e b.\n");
     ler("%f %f",&a, &b);
     PulaLinha
     ImprimindoMensagem(A razao entre eles e:)
     PulaLinha
     ImprimindoRazao(a/b,a,b)
}
```

12.3. Macros Pré-Definidos

- O padrão ANSI define cinco macros intrínsecas:

- __LINE__ : É expandida para o número da linha do arquivo fonte na qual é invocada.
- __FILE__ : É expandida para o nome do arquivo na qual é invocada.
- __TIME__ : É expandida para o instante da compilação do programa.
- __DATE__ : É expandida para a data da compilação do programa.
- __STDC__ : É expandida para uma constante igual a 1 se o compilador está em conformação com o padrão ANSI.

Exemplo 12.2:

```
/* Macros.c: Ilustra o uso de algumas macros pré-definidas */

#include <stdio.h>

void main()
{
     printf("O programa %s foi compilado na data %s, no instante %s.\n",
             __FILE__, __DATE__, __TIME__);
     printf("Agora estamos na linha %d.\n", __LINE__);
}
```

12.4. A Diretiva #undef

- Uma vez que uma macro seja definida, ela retém seu significado até o final do programa ou até que seja removida explicitamente com a diretiva #undef.
- A remoção de uma macro é necessária quando se deseja redefini-la.

Exemplo:

```
#define COMPRIMENTO 100
int matriz1[COMPRIMENTO];

#undef COMPRIMENTO

#define COMPRIMENTO 50
int matriz2[COMPRIMENTO];

 ...
```

12.5. Diretivas de Compilação Condicionais

- Permitem que se compilem porções de código do programa seletivamente.

12.5.1. As diretivas #if, #elif, #else, #endif

Sintaxe:

#if condição_1
    bloco_de _instruções_1
#elif condição_2
    bloco_de _instruções_2
...
#elif condição_n
    bloco_de _instruções_n
#else
    bloco_padrão_de_instruções
#endif

Como funciona:

- Ao encontrar uma diretiva **#if**, o compilador avalia a condição correspondente.

- Se esta condição for verdadeira, as instruções subseqüentes são executadas.
- Caso contrário, as condições associadas a cada diretiva **#elif** são avaliadas, pela ordem, até que uma seja verdadeira.
- Caso isto não ocorra, são executadas as instruções correspondentes à diretiva **#else**, se esta estiver presente.

- Somente um bloco de instruções é compilado dentro de um conjunto **#if** ... **#endif**.

Exemplo:

```
#if SISTEMA == SYSV
    #define CABEC "sysv.h"
        #elif SISTEMA == BSD
            #define CABEC "bsd.h"
        #elif SISTEMA == MSDOS
            #define CABEC "msdos.h"
#else
    #define CABEC "default.h"
#endif

#include CABEC
```

12.5.2. As diretivas #ifdef, #ifndef

- Estas diretivas significam "se definido" e "se não definido" respectivamente.

- Um uso comum destas diretivas é feito com o objetivo de assegurar que o conteúdo de um arquivo cabeçalho seja incluído apenas uma vez, a fim de se evitar que o compilador acuse um erro devido a uma múltipla inclusão. Múltiplas inclusões podem ocorrer facilmente, quando se faz uso de #include aninhados.

Exemplo:

```
#ifndef MSDOS
#define MSDOS
. . .

/* conteúdo de msdos.h aqui */
. . .

#endif
```

Exemplo 12.3:

```
/* Ifdef.c: Ilustra o uso das diretivas de compilação condicional */
#include <graphics.h>
#include <stdlib.h>
#include<stdio.h>
#include <conio.h>

void InitGraf(void);
void main()
{     int midx, midy;
      int raio;  /* Raio do circulo */
      int anguloi, angulof, raiox, raioy; /* Dimensoes da elipse */
      int esquerda, emcima, direita, embaixo; /* Coordenadas para o retangulo */

#define CIRCULO  1
#define ELIPSE   2
#define QUADRADO 3

#ifndef FIGURA
#define FIGURA QUADRADO
#endif

   printf("FIGURA vale: %d\n",FIGURA);
   puts("Pressione qualquer tecla para ve-la.");
   getch();
   InitGraf();
   midx = getmaxx() / 2;
   midy = getmaxy() / 2;
   setcolor(getmaxcolor());
```

```
#if FIGURA==CIRCULO
   raio = 100;
   /* Desenha o circulo */
   circle(midx, midy, raio);
#elif FIGURA==ELIPSE
   /* Desenha a elipse */
   anguloi = 0;
   angulof = 360;
   raiox = 100;
   raioy = 50;
   ellipse(midx, midy, anguloi, angulof, raiox, raioy );
#else
   esquerda = midx - 50;
   emcima = midy - 50;
   direita = midx + 50;
   embaixo = midy + 50;
   /* Desenha o quadrado */
   rectangle(esquerda, emcima, direita, embaixo);
#endif

#ifdef FIGURA
#undef FIGURA
#endif

#define FIGURA CIRCULO
   getch();
   closegraph();
   printf("Agora FIGURA vale: %d\n",FIGURA);
   puts("Pressione qualquer tecla para sair.");
   getch();
}
void InitGraf(void)
{
   /* Requisita auto-detecao */
  int gdriver = DETECT, gmode, errorcode;

  /* Inicializa modo grafico e variaveis locais  */
   initgraph(&gdriver, &gmode, "\\borlandc\\bgi");
      /* Le resultado da inicializacao */
   errorcode = graphresult();
   if (errorcode != grOk)  /* Ocorreu um erro  */
   {
      printf("Erro Grafico : %s\n", grapherrormsg(errorcode));
     printf("Pressione qualquer tecla para sair:");
      getch();
      exit(1); /* Termina com um codigo de erro  */
   }
```

12.6. A Diretiva #line

- Faz com que o compilador acredite que o número de linha da próxima linha seja dado pela constante decimal inteira que se segue a esta diretiva.

Exemplo:

```
#line 100             // inicializa o contador de linhas
void main(void)       // linha 100
{                     // linha 101
  printf("Linha corrente: %d\n", __LINE__); // linha 103

 }
```

12.7. A Diretiva #error

- Permite que se reportem erros durante o estágio de pré-processamento.

Sintaxe:

#error mensagem

- Quando a diretiva é encontrada a mensagem de erro é mostrada.

12.8. A Diretiva #pragma

- Esta diretiva executa tarefas específicas de implementação.

- Cada compilador é livre para suportar nomes especiais que têm comportamento definido pela implementação quando precedido por **#pragma**.

Lista de Exercícios 12:

*1. Defina macros que calculam o quadrado, o cubo e a quarta potência de um valor a ser fornecido pelo usuário. Defina cada uma delas, usando a definição da macro anterior. Exiba o resultado dos cálculos usando macros que utilizem o operador **#** para imprimir mensagens. Remova a definição das macros ao final do programa.*

2. Use as diretivas de compilação condicionais para escrever um programa que imprima ou o quadrado, ou o cubo, ou a quarta potência de um número, conforme seja definida a macro POTÊNCIA. Use as macros definidas no programa do exercício 1 para o cálculo das potências.

Bibliografia

SCHILDT, H. **C Completo e Total**. São Paulo,:Makron Books, 1997.

MIZRAHI, V. V. **Treinamento em Linguagem C - Módulo I e II**. São Paulo: Makron Books, 1990.

CALVERT, C. **Programando Aplicações em Windows com C e C++.** Rio de Janeiro: Berkeley, 1994.

GOTTFRIED, B. S. **Programando em C**. São Paulo: Makron Books, 1993.

KERNIGHAN, B. W., RITCHIE, D. M. **C- A Linguagem de Programação Padrão ANSI**. Rio de Janeiro: Campus.

AITKEN, P., JONES, B. **C - Guia do Programador**. Rio de Janeiro: Berkeley Brasil, 1994.

SCHILDT, H. **Turbo C Avançado – Guia do Usuário**. São Paulo,:Makron Books, 1990.

Esta apostila foi elaborada pela Profa. Dra. Elisamara de Oliveira
Qualquer sugestão/crítica deve ser enviada para bytepapo@ajato.com.br e será muito bem recebida!